9

ADE100302

NCS TIB 79-9

NATIONAL COMMUNICATIONS SYSTEM

TECHNICAL INFORMATION BULLETIN
79-9

MEASUREMENT OF COMPRESSION FACTOR AND ERROR SENSITIVITY FACTOR OF FIVE SELECTED TWO-DIMENSIONAL FACSIMILE CODING TECHNIQUES

SEPTEMBER 1979

DISCLAIMER NOTICE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18 NCS, SBIE

19

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER NCS TIB-79-9, AD-E100 302	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) 6 Measurement of Compression Factor and Error Sensitivity Factor of Five Selected Two-Dimensional Facsimile Coding Techniques.	5. TYPE OF REPORT & PERIOD COVERED 9 Final rept.,
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Neil Randall Richard Schaphorst Steve Urban	8. CONTRACT OR GRANT NUMBER(s) 15 DCA100-79-C-0031 New
9. PERFORMING ORGANIZATION NAME AND ADDRESS Delta Information System, Inc. 259 Wyncote Road Jenkintown, PA 19046	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS National Communications System Office of Technology and Standards (NCS-TS) Washington, D.C. 20305	12. REPORT DATE 11 September 1979
	13. NUMBER OF PAGES 304
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 308	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited; approved for public release

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Coding
Digital Facsimile
Facsimile Coding
Compression Factor
Error Sensitivity
Two-dimensional Coding
Coding Algorithms
Computer Simulation
Image Statistics
CCITT Standards

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Technical Informations Bulletin (TIB) describes the measurement of Compression Factor and Error Sensitivity Factor of five different two-dimensional compression algorithms for use in digital facsimile systems. The five different coding techniques have been proposed to the CCITT by the following five organizations: Japan, IBM, 3M, AT&T, and Xerox. The report contains detailed flow charts and code listings for each of the five computer programs. Compression, error sensitivity, and statistical data has been tabulated.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411 214

NCS TECHNICAL INFORMATION BULLETIN 79-9

MEASUREMENT OF COMPRESSION FACTOR AND ERROR SENSITIVITY FACTOR OF FIVE SELECTED TWO-DIMENSIONAL FACSIMILE CODING TECHNIQUES

SEPTEMBER 1979

PREPARED BY:

DENNIS BODSON
Senior Electronics Engineer
Office of NCS Technology
and Standards

APPROVED FOR PUBLICATION:

Marshall L. Cain

MARSHALL L. CAIN
Assistant Manager
Office of NCS Technology
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program which is an element of the overall GSA Federal Standardization Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee, identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronic Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin presents an overview of an effort which is contributing to the development of compatible Federal, national, and international standards in the area of digital facsimile standards. It has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager
National Communications System
ATTN: NCS-TS
Washington. D.C. 20305
(202) 692-2124

A 33

MEASUREMENT OF COMPRESSION FACTOR AND ERROR SENSITIVITY FACTOR OF FIVE SELECTED TWO-DIMENSIONAL FACSIMILE CODING TECHNIQUES

September, 1979

FINAL REPORT

Submitted to:

NATIONAL COMMUNICATIONS SYSTEMS
8th & S. COURTHOUSE RD.
ARLINGTON, VIRGINIA 22204

CONTRACTING AGENCY:

DEFENSE COMMUNICATIONS AGENCY

Purchase Order: DCA 100-79-C-0031

Submitted by:

DELTA INFORMATION SYSTEMS, INC.
259 WYNCOTE ROAD
JENKINTOWN, PENNA 19046

TABLE OF CONTENTS

APPENDICES

A. Japan CCITT Contribution - No. 42

B. 3M CCITT Contribution - No. 74

C. IBM CCITT Contribution - No. 64

D. XEROX CCITT Contribution - No. 84

E. AT&T CCITT Contribution - No. 81

F. Subroutines which are Common to all Algorithms

G. Flow Chart - Japan Algorithm

H. Code Listing - Japan Algorithm

I. Flow Chart - 3M Algorithm

J. Code Listing - 3M Algorithm

K. Flow Chart - IBM Algorithm

L. Code Listing - IBM Algorithm

M. Flow Chart - XEROX Algorithm

N. Code Listing - XEROX Algorithm

O. Flow Chart - AT&T Algorithm

P. Code Listing - AT&T Algorithm

1.0 INTRODUCTION

Several organizations have submitted contributions to the CCITT (see Appendices A, B, C, D, and E) describing two-dimensional coding techniques for selection of a standard compression algorithm for advanced digital facsimile systems. At the December 1978 meeting in Geneva, a working party of CCITT Study Group XIV adopted specific procedures to measure compression and error sensitivity so that candidate coding techniques may be compared on a meaningful basis. These definitions and procedures are outlined in references 1 and 2. The National Communications System of the U. S. Government has issued three contracts to Delta Information Systems, Inc. to evaluate seven candidate two-dimensional coding techniques using the criteria recommended by the CCITT.

In the first contract (Purchase Order DCA-79-M-0105), a basic computer program was developed to measure the compression and error sensitivity of digital facsimile coding techniques. To validate this program, the Modified-Huffman code, recommended as the one-dimensional standard for Group 3 machines, was tested and simulated on the model. The computer program and work accomplished on this initial contract is described in a Final Report issued August 10, 1979 (see Reference 3).

The document contained herein is the final report describing the work performed under the second contract (Contract DCA 100-79-C-0031). On this program, the validated computer model was used to measure the compression and error sensitivity of five two-dimensional coding techniques. The five coding algorithms selected for simulation are described in the CCITT Contributions which have been reproduced in the appendices listed below.

Appendix	Source of CCITT Contribution
A	Japan
B	3M Company
C	IBM Europe
D	Xerox
E	AT&T

The coding techniques listed above were selected simply because no other contributions had been submitted to the CCITT when this NCS measurement contract was initiated. Contributions were subsequently submitted to the CCITT by the Federal Republic of Germany and the United Kingdom (References 4 and 5, respectively). The NCS organization has issued a third contract (Purchase Order DCA 100-79-M-0209) to Delta Information Systems to measure the compression and error sensitivity of these latter two coding techniques and the results will be issued in a report in October, 1979 (Reference 6).

The measurement parameters which were involved in this program are summarized in Section 2.0 of this report. Section 3.0 describes the hierarchy and interrelationship of computer programs which are used in the measurement process. In many instances, the proposed operation of the coding algorithm was not totally defined when a transmission error was encountered. Section 4.0 describes the generalized error detection and correction procedure which was employed on all algorithm simulations. As the computer programs were prepared for each algorithm, certain assumptions were made for each coding technique, particularly in the area of error detection and correction. These assumptions made for each individual coding technique are documented in Section 5.0.

Twenty separate computer runs were implemented for each algorithm at different combinations of test document, error phase, transmission error file, minimum scan line time, vertical resolution and K-factor. Section 6.0 summarizes the results of these measurements in terms of compression data, error sensitivity data, and coded line length statistics. Section 7.0 contains a list of reference documents related to the contract.

The CCITT contributions describing each coding algorithm have been included in Appendices A through E for reference purposes. Appendix F contains the program code listings for those subroutines which are common to all algorithms, e. g. data packing, data unpacking, error measurement, etc. The remaining ten appendices, G through P, contain the flow charts and the listing of the code for the computer program for each of the five algorithms.

Delta Information Systems wishes to acknowledge the Contracting Officer's Technical Representative, Dennis Bodson, for the extraordinary level of support he has provided during the course of this contract. The assistance of Marla Thomas, from the DCEC computer facility, is also greatly appreciated.

2.0 MEASUREMENT PARAMETERS

In this section, the various parameters involved in the measurement of compression and error sensitivity will be summarized. In general, Study Group XIV of the CCITT agreed upon these measurement parameters at the general meeting held in Geneva in December 1978 (see Reference 2).

2.1 Test Documents

The test documents were chosen from the eight CCITT test documents (see Figure 2-1) since they have been widely used by data compression experimenters in the past. Documents numbered 1, 4, 5, and 7 (see Figures 2-2, 2-3, 2-4, and 2-5 respectively) were selected as the standard test images since these were considered most representative of documents to be transmitted.

The French PTT Administration has scanned the eight CCITT documents at the high resolution specified for Group 3 machines--7.7 lines/mm. They have also quantized each pel to be either black or white and stored the resultant image on magnetic tape. This tape was used as the source of input documents in this simulation program. Appendix B of Reference 3 describes the format of the test document magnetic tape supplied by the French PTT.

2.2 Resolution

It was agreed that measurements would be performed at both standard resolution (3.85 lines/mm.) and high resolution (7.7 lines/mm.). In the high resolution case, all lines on the input test documents shall be used. In standard resolution tests, every odd scan line

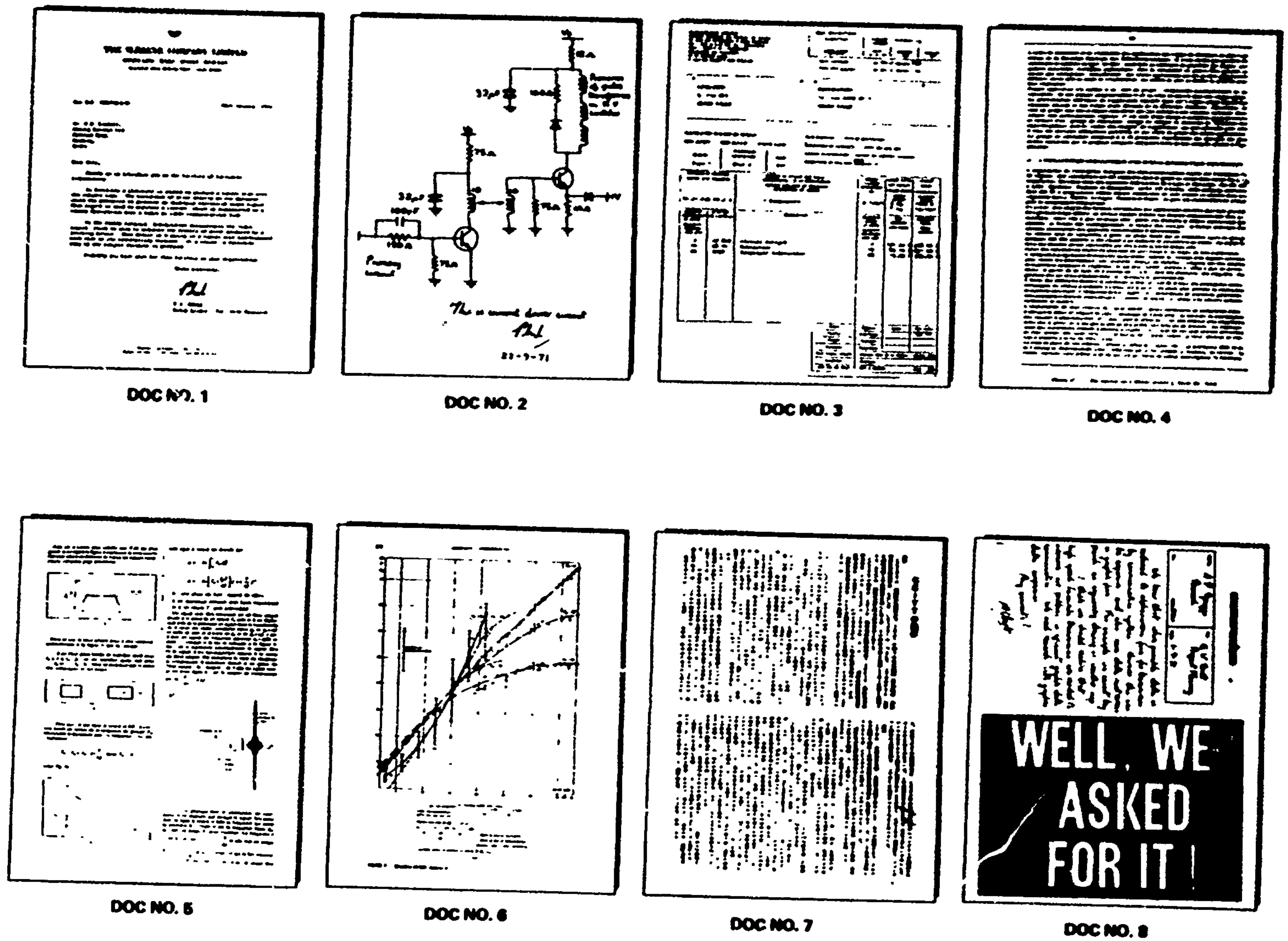


Figure 2-1 CCITT Standard Test Documents

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH 25 8 ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC 18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Figure 2-2 CCITT Test Document No. 1

Registered in England: No. 2038
Registered Office: 60 Vicars Lane, Ilford. Essex.

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet,et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 30 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 300 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger,et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gérera" environ un million d'abonnés à la fin du VIème Plan
La mise en place de ces centres a débuté au début de l'année 1971 un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Figure 2-3 CCITT Test Document No. 4

Photo n° 1 - Document très dense lettre 1,5mm de haut -

Restitution photo n° 9

Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

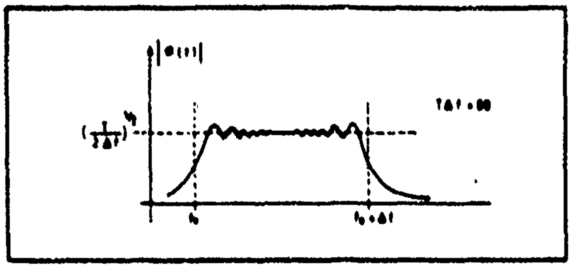


FIG. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leq f \leq f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

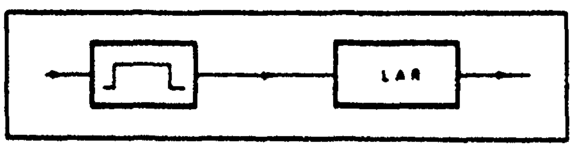


FIG. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_R décroissant linéairement avec la fréquence f suivant l'expression :

$$T_R = T_0 + (f_0 - f)\frac{T}{\Delta f} \quad (\text{avec } T \sim T)$$

(voir fig. 4).

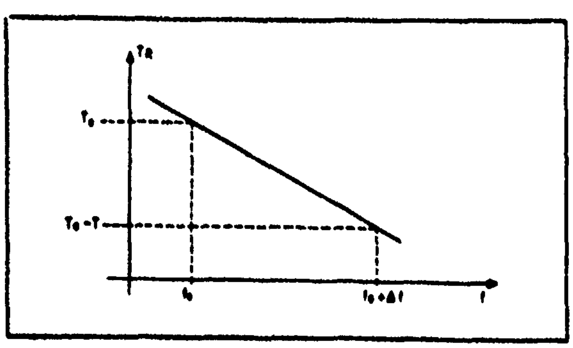


FIG. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi \int_0^f T_R \, df$$

$$\varphi = -2\pi\left[T_0 + \frac{f_0 T}{\Delta f}\right] f + \pi \frac{T}{\Delta f} f^2$$

Et cette phase est bien l'opposé de $/\phi(f)$,

à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de récepteur à compression d'impulsion donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\frac{T}{1/\Delta f} = T\Delta f$

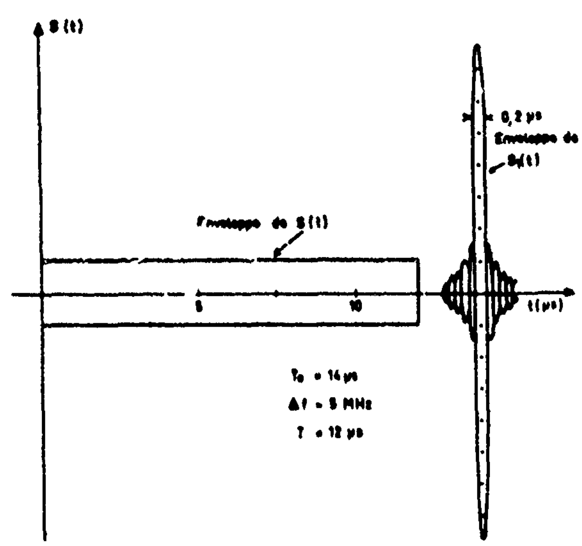


FIG. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\frac{T}{\Delta f}$ et elle met un temps $T_0 - (f-f_0)\frac{T}{\Delta f}$ pour traverser, ce qui la fait ressortir à l'instant T_0 également. Ainsi donc, le signal $S(t)$

Figure 2-4 CCITT Test Document No. 5

CCITTの概要

沿革

CCITTは、国際電気通信連合（ITU）の四つの常設機関（事務総局、国際周波数登録委員会、CCIR、CCITT）の一つとして、ITUの中でも、世界の国際通信上の諸問題を真先に取上げ、その解決方法を見出して行く重要な機関である。日本名は、国際電信電話諮問委員会と称する。

CCITTの前身は、CCIF（国際電話諮問委員会）とCCIT（国際電信諮問委員会）である。CCIFは、1924年にヨーロッパに「国際長距離電話通信諮問委員会」が設置され、これが1925年のパリ電信電話会議のとき、正式に、「国際電話諮問委員会」として万国電信連合の公式機関となったものである。CCITは、同じく1925年の会議のとき、CCIFと併立するものとして設置された。

そして、CCIFは、1956年の12月に第18回総会が開催されたのち、CCITは、同年同月に第8回総会が開催されたのち、併合されて現在のCCITTとなった。このCCITTは、CCIFとCCITが解散した直後、第1回総会を開催し、第2回総会は、1960年にニューデリーで、第3回総会は、1964年、ジュネーブで、第4回総会は、1968年、アルゼンチンで開催された。

CCIFとCCITが合併したのは、有線電気通信の分野、とくに伝送路について電信回線と電話回線とを技術的に分ける意味がなくなってきたこと、各国とも大体において、電信部門と電話部門は同一組織内にあること、CCIFの事務局とCCITの事務局の合併による能率増進等がおもな理由であった。

CCITTは、上述のように、ヨーロッパ内の国ぐにによって、ヨーロッパ内の電信・電話の技術・運用・料金の基準を定め、あるいは統一をはかってきたので、現在でも、その影響を受け、会合参加国は、ヨーロッパの国が多く、ヨーロッパで生起する問題の研究が多い。たとえば、1960年のCCITT勧告の中で、技術上配慮する距離は約2,500kmであったが、これはヨーロッパ内領域を想定したものである。

しかしながら、1956年9月に敷設された大西洋横断電話ケーブルは、大陸間電話通信の自動化および半自動化への技術的可能性を与え、CCITTがこの問題を取り上げるに及び、CCITTの性格は漸次、汎世界的色彩を実質的に帯びるに至った。この汎世界的性格は第2次世界大戦後目ざましくなったアジア・アフリカ植民地の独立に伴ってITUの構成員の中にこれらの国が加わり、ITUの中に新しい意見が導入されてきたことにもあらわれている。CCITTの汎世界化は、1960年の第2回総会がニューデリーで開催されたことにもあらわれている。この総会までは、CCIT、CCIFのいずれにしろ、アメリカやアジアで総会が開催されたことがなく、CCITT委員長も、ニューデリー総会の準備文書で、この点には注目すべきであるとのべている。

任務

ITUは、全権委員会議、主管庁会議を始めとして、七つの機関をもち、それぞれの機関の権限と任務は国際電気通信条約に明記されている。そこで条約を参照してみるならば、CCITTの任務は、つぎのとおりとなっている。

「国際電信電話諮問委員会(CCITT)は、電信および電話に関する技術、運用および料金の問題について研究し、および意見を表明することを任務とする。」（1965年モントルー条約第187号）

「各国際諮問委員会は、その任務の遂行に当たって、新しい国または発展の途上にある国における地域的および国際的分野にわたる電気通信の創設、発達および改善に直接関連のある問題について研究し、および意見を作成するように妥当な注意を払わなければならない。」（同第188号）

「各国際諮問委員会は、また、関係国の要請に基づき、その国内電気通信の問題について研究し、かつ、勧告を行なうことができる。」（同第189号）

上記第187号と第188号にいわれる「意見」とは、フランス語の Avis から訳したもので、英語では、「勧告(Recommendation)」となっている。CCITTの表明する意見は、国際法的には強制力をもたないものであって、この点が、条約、電信規則、電話規則等各国を拘束する力をもっているものと異なる。もっとも意見とは称しても、技術的分野では、電信規則のごとき、各国政府が承認してその内容を実施する強制規則をもたないので、実際にある機器の仕様を定める場合には、多くの国の意見が統一されたこの「意見」に従わなければ、円滑な国際通信を行なうことができない場合が多い。この意見（または勧告）は、国際通信を行なう場合各国が直面する問題について、具体的意見を表明するもので、たとえば、大陸間ケーブルで大陸間通話を半自動化しようとする場合、その信号方式や取り扱う通話の種類および料金は、どのようにするかを研究して意見を表明する。したがって、CCITTの活動は、つねに時代の最先端を行くもので、CCITTの活動方向は、そのまま世界の国際通信の活動方向であるともいえる。

この意見は、また、電信規則以下のその他の規則のごとく、数年以上の間隔をもって開催される主管庁会議というような大会議の決定をまたなくても表明することができ、また、その改正も容易であるので、現在のように進歩の早い国際通信界では、関係国の意見を統一した国際的見解としては非常に便利である。

Figure 2-5 CCITT Test Document No. 7

should be used. Figure 2-6 is a copy of the French PTT Test Document No. 4 scanned with 7.7 lines/mm. resolution. Figure 2-7 is a copy of the same document where the even scan lines have been replaced with the line above. Therefore, this represents a document in which the vertical resolution is 3.85 lines/mm.

2.3 Minimum Scan Line Time (MSLT)

The standard MSLT to be used in the measurement program will be 5, 10, and 20 ms. with EOL-code and 0 ms. without EOL-code. It was later clarified in a memo from the chairman of the Working Committee (see Reference 7) that if, for reasons of test economy, only one value of MSLT can be used in the test program, that value shall be 20 ms.

2.4 Transmission Bit Rate

The standard transmission bit rate is 4800 bits/sec.

2.5 Measurement of Compression

Two standard measures of compression have been established-- (1) number of coded bits (2) Compression Factor. The number of coded bits is the number of bits required to transmit a document, including all overhead bits such as End of Line (EOL) and Fill bits. The Compression Factor is computed by dividing the total number of picture elements (pels) per test document by the number of coded bits. It was further agreed that the Compression Factor and coded bits should be computed for two different conditions--with overhead and without overhead. The measurement with overhead applies to the

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation [illegible] se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet, et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger, et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux,

Figure 2-6 Test Document Scanned/Printed 7.7 lines/mm.

Photo n° 1 - Document très dense lettre 1,5mm de haut -

Restitution photo n° 9

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet,et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger,et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Figure 2-7 Test Document Scanned 7 lines/mm. Printed 3.85 lines/mm.

Photo n° 1 - Document très dense lettre 1,5mm de haut -

Restitution photo n° 9

Group 3 situation while the measurement without overhead applies to the Group 4 case.

2.6 Measurement of Error Sensitivity

An objective measure of error sensitivity is obtained by encoding the test documents with the proposed techniques (all overhead bits must be included), subjecting the resulting bit stream to transmission errors, decoding the transmission to obtain the received image, and comparing the original image with the received image to determine the number of pels in error. The Error Sensitivity Factor (ESF) is calculated as the total number of document pels in error divided by the total number of transmission bits that are in error. In this way, the ESF represents the average disturbance to the output image caused by a single transmission error.

2.6.1 Transmission Error Pattern

It was agreed that a record of actual bit errors incurred over telephone lines will be used in the error sensitivity test. The Federal Republic of Germany (see Reference 8) has obtained a record of such errors by transmitting a known psuedo-random sequence at 4800 bits/sec. using a V27 ter modem over a switched telephone network. The resultant error pattern has been recorded on magnetic tape and made available to experimenters. Appendix C of Reference 3 describes the format of the transmission error magnetic tape. This tape was used in the measurement of error sensitivity described in this report.

2.6.2 Error Phases

One concern with the ESF measurement is the high degree of sensitivity to those few errors which may affect the end of line code and can cause an inordinate number of incorrect pels. If the error pattern happened to fall in an unfortunate phase relative to the encoded bits, a large number of pels could be affected. On the other hand, the error pattern could fall fortuitously and affect a relatively few number of pels. To insure experimenters can achieve an adequate level of statistical validity, the concept of error phases has been introduced. In the basic zero phase, the first bit of the error record is aligned with the first bit of the encoded transmission. In the case of Phase 2, the transmitted bit information is delayed by 1,024 bits relative to the previous run. The transmission bit information is delayed by 2,048 bits for Phase 2. Experimenters would have a higher confidence level in the average of the three phases compared to any one ESF taken alone.

2.6.3 Error Correction

In order to precisely measure the error sensitivity, both the encoding technique and the decoding algorithm must be completely defined. If more than one decoding algorithm is proposed (for example, to achieve differing levels of error control), each must be tested separately. Collective Letter No. 87 from the CCITT (see Reference 7) outlines an error correction procedure to be used for simulating two-dimensional algorithms where an error correction procedure has not been otherwise specified. In this procedure, the erroneous line is replaced

by the previous line and following lines are replaced by white lines until a one-dimensional coding line is correctly decoded.

3.0 COMPUTER PROGRAM OVERVIEW

This section contains a general overview of the computer program architecture written under this contract. The description is divided into two parts. Section 3.1 focuses on the overall simulation process from a flow perspective with particular emphasis on the simulation inputs and outputs. Section 3.2 presents the hierarchical structure of the programs illustrating how the programs are organized for each of the 5 different algorithms. For convenience of the reader, a detailed flow chart, and the actual program code listing, has been included in the Appendices for each algorithm (Appendices F through P). All computer programs have been written in conventional Fortran IV language.

3.1 The Simulation Process

Figure 3-1 illustrates the interrelationship between the major functions of each simulation program developed on the subject contract. There are two input data sets to each simulation which originate on magnetic tape. One tape, supplied by the French PTT Administration, contains all eight of the CCITT test documents. The format of this input image tape is described in Appendix B of Reference 3. The other tape, supplied by the Federal Republic of Germany, contains transmission error data from actual switched telephone circuits. The format of this input tape is described in Appendix C of Reference 3. A program called "REDTAP" was prepared to read the data from the input document tape while the error tape is read in directly. Data from the two input tapes are placed on disc in the computer system to be accessed during the simulation process. A separate file is established for each of the

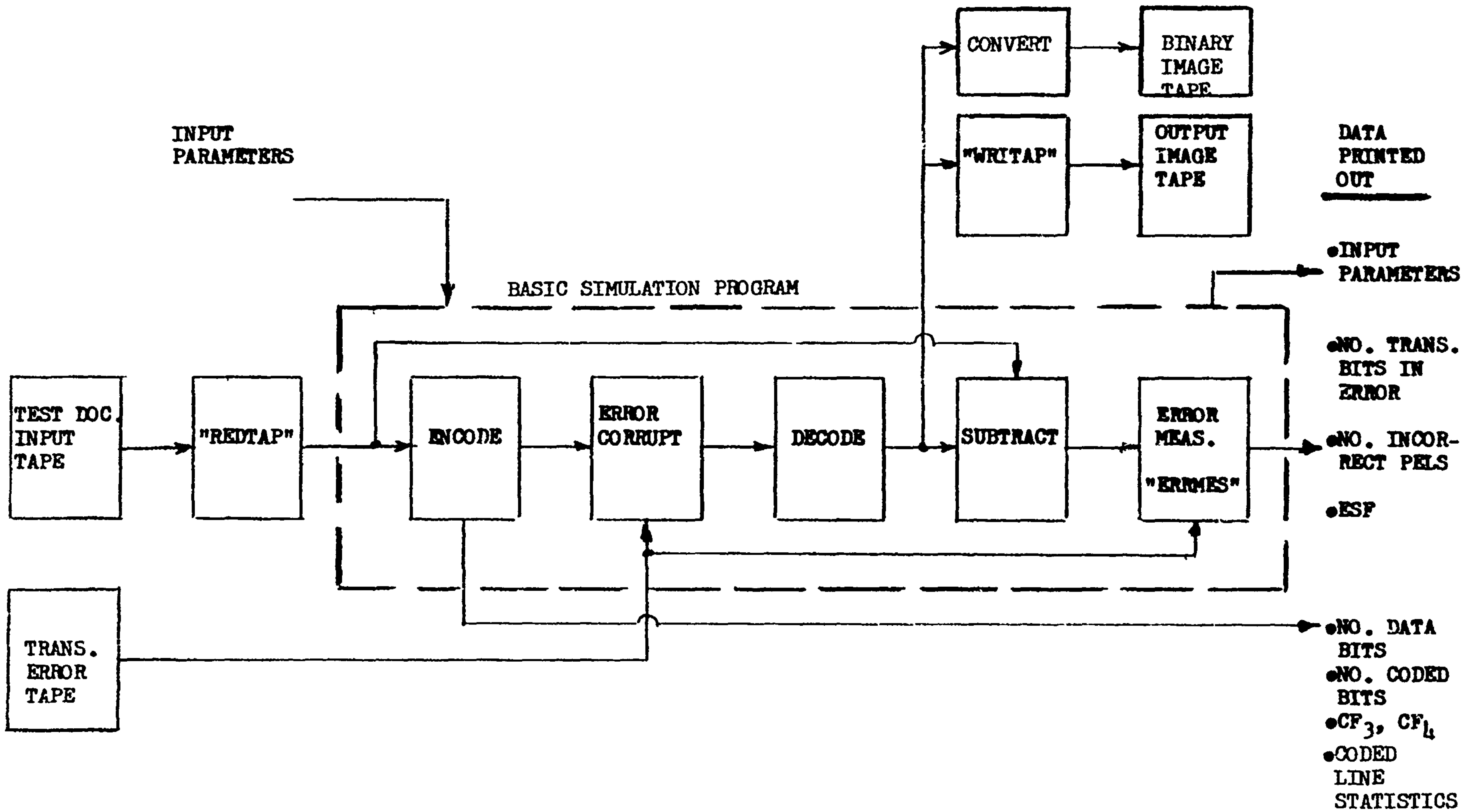


FIGURE 3-1 DIAGRAM OF THE SIMULATION PROCESS

test documents. The transmission error tape is divided into four files, one for each of four different circuit error conditions.

To initiate the simulation process, the operator must type in a set of input parameters. The insertion of the input parameters is accomplished on an interactive basis with prompting. A typical interactive sequence with responses is listed below.

1. PARAMETERS: INPUT (=I), OR DEFAULT (=D)? I
2. DIAGNOSTIC PRINTOUT? (Y OR N). N
3. ENTER MAXIMUM NUMBER OF PELS PER LINE: 1728
4. ENTER VERTICAL SAMPLING: 1
5. ENTER PARAMETER K: 4
6. ENTER ERROR PATTERN PHASE: 0
7. ENTER MINIMUM COMPRESSED LINE LENGTH: 96
8. NUMBER OF SCAN LINES TO BE PROCESSED = ? 10
9. ERROR MODE = ? (M=MANUAL, T=TAPE, N=NO ERRORS) N

After the data has been entered and the measurement parameters have been selected, the first step in the simulation process is the "ENCODE" function. This function detects color changes in the input data and constructs the appropriate code word by table look-up or algorithm. The actual code is fed to the error corrupt unit, while the number of code bits is accumulated with fill and EOL codes to provide the output total number of data bits, to compute the Compression Factors, CF_3 and CF_4.

The error corruption step combines the transmission error data with the encoded data. At each point in the image where an error occurs, the corresponding bit in the encoded signal is reversed and fed to the

decode function. The decoder basically performs the inverse function of the encoder, generating a series of lines of image pels. There are two parts of the decoding function which are not obvious and require clarification: (1) what the decoder does when an error occurs (2) what the decoder does when a line is missing. The operation of the decoder under these two conditions is described in Section 4.

The output of the Decode function feeds the "WRITAP" or "CONVERT" functions for writing the error corrupted image on magnetic tape. It is also fed to a subtraction function which compares the decoded image with the original image. Pels which are in error are fed to the "ERRMES" subroutine which counts all the pels in the image which are in error. This subroutine also counts the number of transmission error bits which corrupted the encode signal. Finally, the "ERRMES" subroutine computes the ESF by dividing the number of incorrect pels by the number of transmitted bits in error.

Figure 3-1 shows that the simulation process provides a printout of all the computed performance data as well as a summary tabulation of the input parameters.

For more details on the computer programs, refer to Section 3.2 for a description of the program structure and to the Appendices for flow charts and program listings.

The reader should note that most of the software prepared under this contract is suitable for simulating any compression algorithm. The only subroutines which must be written specifically for a particular coding technique are the encode and decode subroutines.

3.2 Program Structure

The following section describes the structure of the computer program written to simulate the various algorithms. In addition, a brief description of each of the subroutines is given.

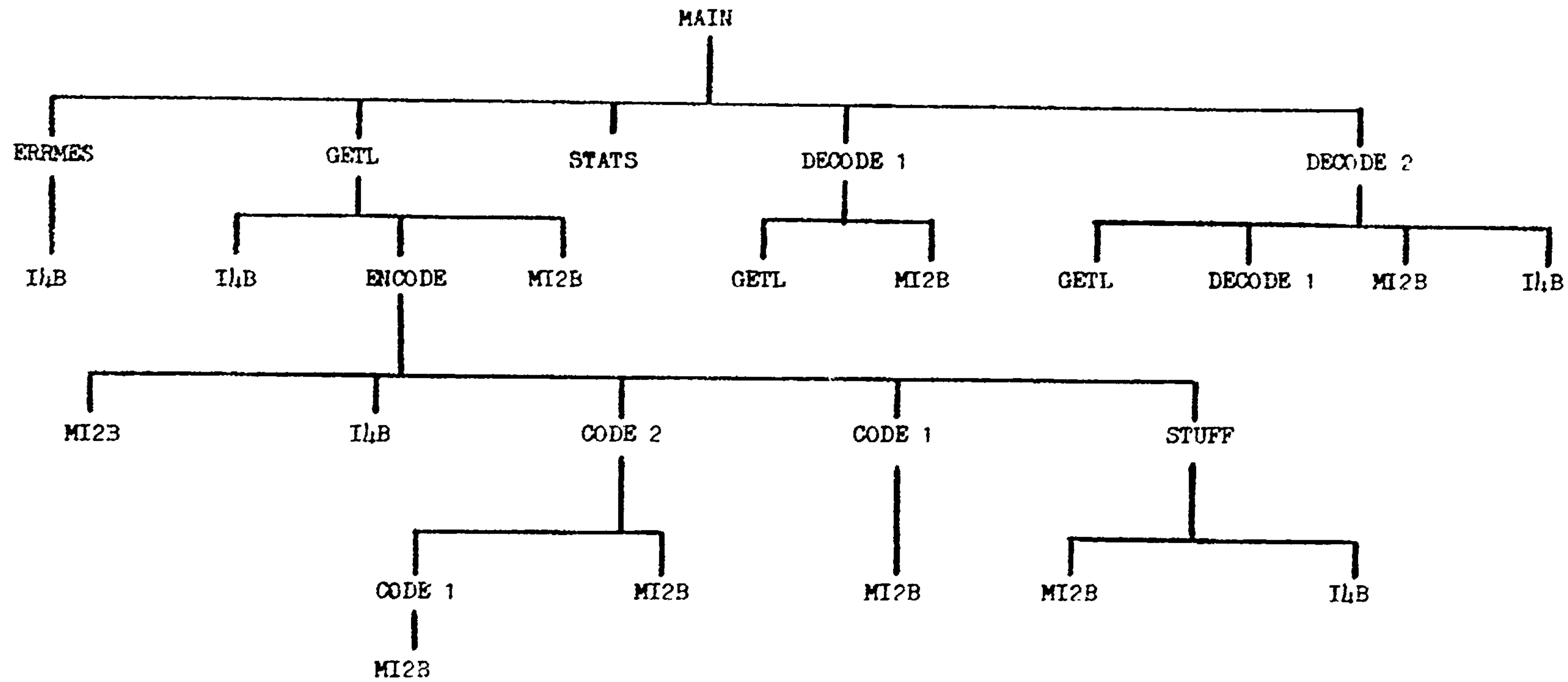
Each of the computer programs written to simulate the five compression algorithms conforms to the general structure shown in Figure 3-2. The chart given in this figure shows the hierarchy of the functions that make up each simulation program. Some of the functions on the chart are named generically: the table in Figure 3-2 shows how these generic function names are keyed to the actual subroutine names used by each compression algorithm. The names on the hierarchical chart that do not appear in the table are subroutines that are used by all compression algorithms. A brief description of each of the functions/subroutines follows:

MAIN

The MAIN program controls the decoding process and the error recovery procedure for getting back in sync when an error is detected. As can be seen from Figure 3-2, the simulation process is "decode driven"; that is, the main program controls the decode process which decodes a buffered line of compressed data. When the contents of the buffer have been used up, a new line of data is encoded. The MAIN program also controls parameter input, measurement of errors, and reports computed results.

GETL

The GETL subroutine retrieves a number of requested bits from



FUNCTION	SUBROUTINE NAMES				
	READ	3M	IBM	XEROX	AT&T
MAIN	JPREAD	THREEM	IBM	XEROX	BTL
GETL	GETL	GETL 3	GETLI	GETLX	GETLB
ENCODE	ENCODR	ENCOD3	ENCODI	ENCODX	ENCODB
CODE 1	CODELN	CODELN	CODELN	XCODLR	CODELN
CODE 2	CODEVH	CODE3M	CODIBM	CODEX	CODBTL
DECODE 1	ONEDIM	ONED3	ONEIBM	ONEXOX	ONEBTL
DECODE 2	TWODIM	TWOD3	TWOIBM	TWOXOX	TWOBTL

FIGURE 3-2 SUBROUTINE HIERARCHY

the coded line and delivers the bits packed into a word (right justified). If stuffing bits have been used, i.e. in the READ code, they are removed. End-of-line codes (EOL) or line synchronization signals (LSS) are detected. If the number of coded bits requested by the calling program is not available, the ENCODE subroutine is called to provide them.

ENCODE

This subroutine supplies a line of compressed data. Color transitions on an input line are detected bit-by-bit. Both one-dimensional and two-dimensional lines are encoded depending on the parameter K. The code word is generated by table look-up, or algorithm, as appropriate, and added to the coded line buffer via CODE 1 and/or CODE 2.

CODE 1

The subroutine CODE 1 is called by ENCODE to look up the Modified Huffman Code (MHC) corresponding to a given run length and color, and add the code word to the coded line buffer.

CODE 2

The subroutine CODE 2 performs a similar function for the two-dimensional case. Based on a particular feature, the appropriate code word is generated by table look-up or algorithm and added to the coded line buffer. All code tables for both one-dimensional and two-dimensional codes are stored in labelled common which is initialized by a BLOCK DATA subprogram.

STUFF

The STUFF subroutine is used only by the READ compression algorithm to insert 0's in the coded data stream in order to avoid ambiguities with the line synchronization signal. A '0' is inserted after every occurence of five consecutive ones in the coded data stream.

DECODE 1

The DECODE 1 subroutine decodes the MHC. It extracts a set of n bits (n=3 initially) from the coded line and looks for a match with all code words of length n, increasing n until a match is found or the code table is exhausted. When and if a match is found, the indicated bits are constructed on the output line. Any errors detected in the decoding process, such as no match to code table, or line too long, are flagged.

DECODE 2

This subroutine performs the same function as DECODE 1 for the two-dimensional line.

MI2B and I4B

The subprograms MI2B and I4B are used to pack and unpack a set of bits into (or from) an array of words.

4.0 Error Detection/Correction Procedure

In Reference 7.0, the following error checking and processing procedure was specified by the CCITT for testing the proposed two-dimensional coding techniques:

1) Error checking - If decoded signals are not exactly 1728 pels/line, the line is recognized as an erroneous line.

2) Error processing - The erroneous line is replaced by the previous line and following lines are replaced by white lines until one-dimensional coding line is correctly decoded.

The error detection and correction procedures used in this simulation follow the spirit, if not the letter, of this directive.

Not all of the proposed algorithms produce a line pel count that can be checked against the correct 1728 pels per line. The error checking was expanded to include the detection of any condition that could not possibly occur in a correctly received transmission. Some examples of possible error conditions are:

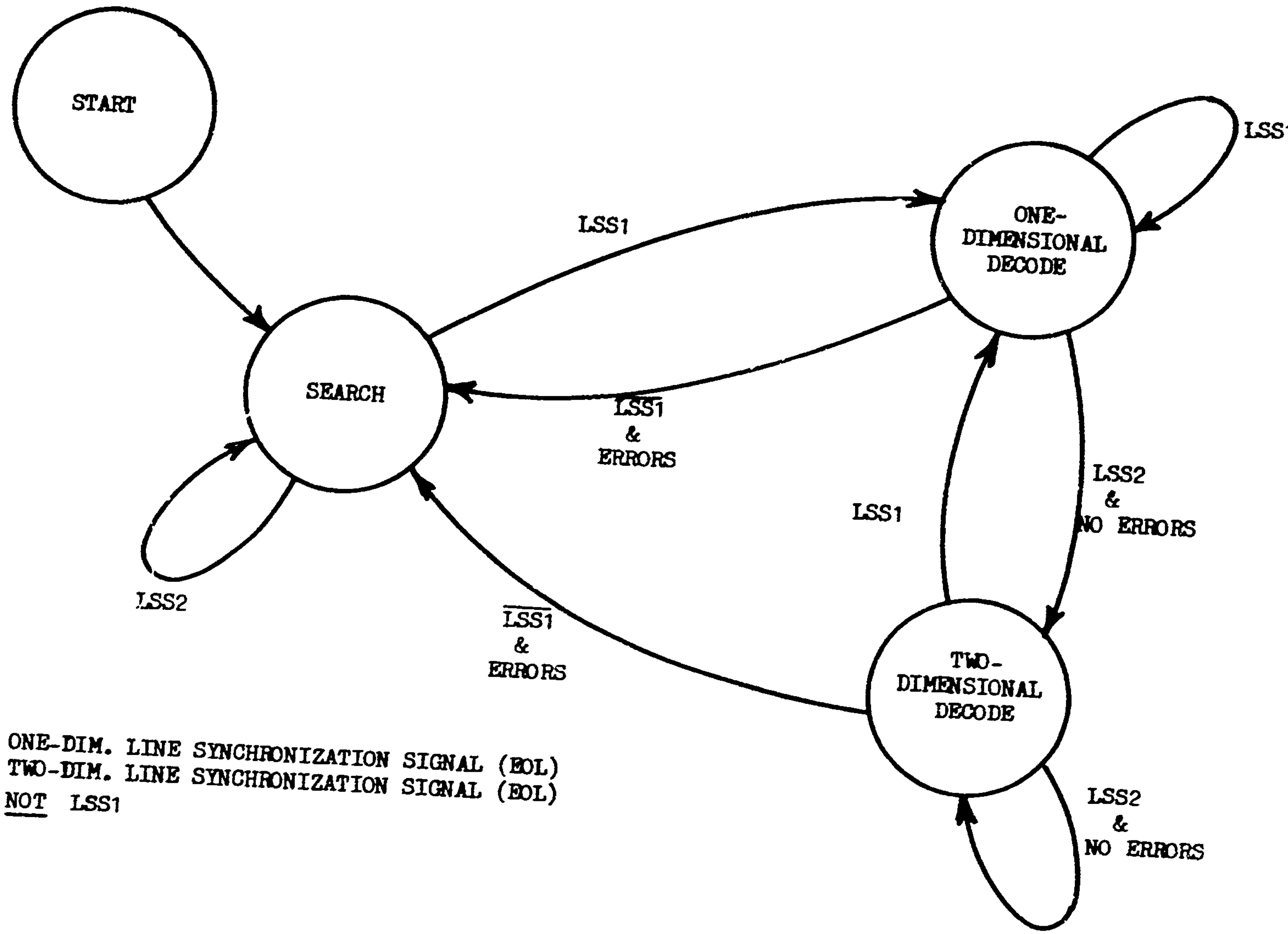
- EOL occurs before 1728 pels have been written
- More than 1728 pels have been written before EOL is received
- No word in applicable code table matches received bit pattern
- Current line decoding references a run that does not exist in the previous line
- Pels are written to the left of the first pel on the line

Conditions that are only improbable, such as a line of pels that differs radically from the previous line, are not considered error conditions. Error conditions specific to each coding algorithm are discussed in Section 5.0.

The AT&T algorithm does not, strictly speaking, have a "one-dimensional coding line." Therefore, the error processing was extended, for this algorithm, to consider any line that can be decoded without an error condition as a correct line. In decoding lines that reference previous lines, the last correctly decoded line is used as the reference line, regardless of whether or not there are intervening error lines. It is believed that the chance of correctly decoding a line, following an error line that references a previous line, is extremely small.

Upon detection of an error condition, the decoder attempts to resynchronize by searching for the next unique Line Synchronization Signal (LSS). All but the AT&T algorithm have different codes for one-dimensional and two-dimensional lines. The state diagram for error recovery for these algorithms is shown in Figure 4-1. For the AT&T algorithm, the One-Dimensional Decode and the Two-Dimensional Decode states are identical, and detection of an EOL in the Search state causes a change to the Decode state, rather than staying in Search.

Following Reference 7, when an error condition is detected, the error line is replaced by the previous correct line, while successive error lines are replaced by all-white lines, until a line is decoded correctly. It should be pointed out that this procedure



LSS1 = ONE-DIM. LINE SYNCHRONIZATION SIGNAL (EOL)
LSS2 = TWO-DIM. LINE SYNCHRONIZATION SIGNAL (EOL)
$\overline{LSS1}$ = <u>NOT</u> LSS1

Figure 4-1 DECODE STATE DIAGRAM

may not be optimum. Repeating the last correct line until the next correct line is received may produce better results from a subjective and objective point of view.

Because of transmission errors, some of the original image lines may be missing in the output, or additional lines may be in the output that were not in the original image. In order that a missing or extra line not have an undue influence on the ESF, it is important that the original and received images not get permanently out of line alignment when they are compared to determine the number of pel errors. To this end, each of the lines in the original image is assigned a serial line number, and this number continues to be associated with the same line in the received image. If a transmitted line is dropped, due to the loss of an EOL, then its line number will be missing in the output. On the other hand, if a line is broken into two or more lines in the received image, due to false EOL's, then its line number will appear more than once in the output.

If no lines are dropped or added, the line numbers of the original and received lines that are compared to detect pel errors will be equal. When a line is added or deleted, the line numbers of the compared lines will become unequal. When this occurs for the first time, the two lines with different line numbers are compared to determine the number of pel errors, which is added to the pel error total. Then, instead of proceeding to the next line in both the original and received images, the next line is used in only one of the images, with the previous line being used in the other image. The line is advanced only in that image that has the smaller line number, so as to tend to make

the line numbers of the two images more equal. This continues until the line numbers are equal, after which the next line is used in both images, until another inequality is detected.

This procedure provides a proper penalty for a missing or added line, but prevents this type of error from causing pel errors over the entire image below the place where it occurred.

5.0 ASSUMPTIONS RELATED TO INDIVIDUAL ALGORITHMS

This section describes any modifications that were made to the compression algorithms that were not covered in the contributions. Assumptions and clarifications are also included.

5.1 Japan Algorithm

No modifications were made to the READ algorithm. However, there was a question concerning the adaptive coding. Specifically, should the decision to use horizontal or vertical mode code be made before or after bit-stuffing is performed? Since the examples in Appendix A showed that this decision was made before bit-stuffing, the same approach was taken in the simulation. Bit stuffing was accomplished after a complete line was encoded.

5.2 3M Algorithm

Two assumptions were made concerning the 3M algorithm. First, the optional PASS mode was not included in the simulation for the following reasons: It was felt that the criteria for using the optional pass mode was not well defined. In addition, the optional pass mode was not included in the verbal description nor in the flow charts of Appendix B, nor was it included in the examples of Figure 4 of the same reference.

The second assumption made concerns the decision as to whether to transmit a one-dimensional or a two-dimensional line. It was assumed that if both coded lines are of the same length, a one-dimensional line will be transmitted. It follows then, that if the one-dimensional line is filled, it will be the line that is transmitted.

During the preparation of the simulation program, an error was found in the code table (Figure 3 of Appendix B). The code for VMZ was given as '0'. Since this is ambiguous with the other code words, a '1' was used for VMZ. After the computer program was completed and all of the picture data had been processed, an addendum was received that corrected the error in a somewhat different fashion: All code bits were complemented so that 0's became 1's and 1's became 0's. The result is that the number of coded bits (and compression factor) for the two techniques is the same, but the effects of errors in the coded data stream may cause slight differences in the error sensitivity.

5.3 IBM Algorithm

No modifications were made to the IBM algorithm. However, Appendix C did not clearly define the treatment of the left and right edges. It was deduced from the reference that the first code word on a line always represents a white run. Therefore, any line that begins with a black pel is coded as a white run of zero length. In order to handle this easily, the transition which starts the first run is assumed to be one element to the left of the first pel.

At the right edge of the document (according to Appendix C) "a hypothetical C is assumed after the rightmost pel." It was assumed that this hypothetical C has the color opposite to the last pel on the line. This assumption loses significance when the algorithm is applied to the French data, since all lines were extended by 48 white pels.

5.4 XEROX Algorithm

The XEROX compression algorithm consists of run length encoding runs of correct predictions followed by an incorrect prediction. However, Appendix D does not seem to cover the case where the last pel on a line is predicted correctly. Therefore, it was assumed that when the last pel on a line is predicted correctly, an incorrect prediction is assumed for the next pel (one past the edge) in order to end the line with an incorrect prediction.

Since the predictor "window" overlaps both left and right edges by two pels, it was assumed that these pels are white.

The reference allows the fallback to one-dimensional coding to be either Modified-Huffman Code or "zeroing out" the reference line of the predictor. The former approach was taken to be consistent with the other compression techniques.

One other modification was necessary in simulating the XEROX code. Normally a concurrent search is carried out for the EOL codes while decoding data. This was not possible with the XEROX code because the short EOL preceding the two-dimensional lines is ambiguous with some one-dimensional code words. This means that if the short EOL is used to signify the end of a one-dimensional line, it may end prematurely, causing an error.

This problem was solved by ending a line only after decoding exactly 1728 pels and then checking for the EOL code.

5.5 A T & T Algorithm

This algorithm, also known as the Frank Code, is described in Appendix E. One minor modification was made to the algorithm.

The AT&T Code will encode an initial all-white line with no code at all between EOL's. Successive all-white lines will also be encoded in the same way. Thus, it is possible to obtain a string of successive EOL's. Since the End of Message (EOM) is signalled by 6 successive EOL's, as in the one-dimensional code, a false EOM could easily be obtained, even without transmission errors. Some decoders may wish to declare EOM with fewer than 6 successive EOL's in order to obtain a more reliable detection of EOM. Therefore, the AT&T Code was changed to provide for a zero fill bit before the EOL if there was no other code between the EOL's. In effect, the minimum number of bits per line is 13, regardless of the specified minimum compressed line length. This extra bit permits the decoder to distinguish between successive EOL's due to all-white lines and the EOM.

The error conditions that can occur with this code are (see Appendix E):

1) black pels are written to the left of the first pel or to the right of the last pel.
2) a code word references a non-existent black run in the previous line
3) two black runs touch or overlap
4) the beginning of a run is to the right of the same run
5) no code word in the code table matches the received bits

Note that receiving an EOL before all 1728 pels have been written is a normal occurence for the Frank Code.

The code table used is that in Appendix B of Appendix E.

6.0 MEASUREMENT RESULTS

During the course of this contract, Delta Information Systems prepared five separate computer programs to simulate each of the 5 coding algorithms. All 5 of these programs were run on the Hybrid Computer Facility at the Defense Communications Engineering Center in Reston, Virginia. Twenty computer runs were performed for each of the 5 coding algorithms. The parameters for these runs and the raw output measurement data for each run is tabulated in section 6.1. Sections 6.2 and 6.3 highlight and summarize the compression and error sensitivity data respectively.

6.1 Raw Measurement Data

The parameters which are varied in the simulation process are listed below along with the different values that each parameter was assigned throughout the 20 test runs.

Test document number	1,4,5,7
Error phase	0,1,2
Transmission error file	1,2,3,4
Minimum scan line time (ms)	10,20
K-factor	2,4
Transmission bit rate (Kbps)	4,800
Resolution (lines/mm)	3.85, 7.7

If every combination of these parametric values was tested, a total of 384 computer runs would be required which is neither reasonable nor necessary. Twenty computer runs were executed, and Table 6-1 is a tabulation of the different parameters for each run. For reasons of

TABLE 6-1 TABULATION OF TEST RUN PARAMETERS

TEST RUN	TEST DOCUMENT NUMBER	ERROR PHASE	TRANSMISSION ERROR FILE	MIN. SCAN LINE TIME (ms.)	VERTICAL RESOLUTION * (lines/mm.)
1	4	0	1	20	3.85
2	4	0	1	20	7.70
3	4	0	2	20	3.85
4	4	0	2	20	7.70
5	4	0	3	20	3.85
6	4	0	3	20	7.70
7	4	0	4	20	3.85
8	4	0	4	20	7.70
9	4	1	1	20	7.70
10	4	2	1	20	7.70
11	4	0	1	10	3.85
12	4	0	2	10	7.70
13	4	0	3	10	3.85
14	4	0	4	10	7.70
15	1	0	1	20	3.85
16	1	0	1	10	7.70
17	5	0	1	10	7.70
18	5	0	1	20	3.85
19	7	0	1	20	3.85
20	7	0	1	10	7.70

* For Resolution of 3.85 lines/mm K=2

" " " 7.70 " K=4

test economy, the set of parameters used in Test Run number 1 has been selected as a baseline, and other runs were chosen as variations from that set of values. Again, for reasons of test economy, the K-factor was set at 2 and 4 when the resolution was chosen to be 3.85 and 7.7 lines/mm. respectively.

The raw test results relating to compression and error sensitivity for each coding algorithm are included in Table 6-2 through 6-6. The definitions of these measurement parameters are reviewed below.

- Coded Data Bits - Total compressed bits required to transmit the document excluding all overhead bits - EOL, fill, etc.
- Coded Bits - Total compressed bits required to transmit the document including all overhead such as EOL, fill, etc.
- CF_4 - Number of document pels* divided by the number of coded data bits
- CF_3 - Number of document pels* divided by the number of coded bits
- BER - Transmitted bits in error divided by the number of coded bits
- ESF - Number of incorrect pels divided by the number of transmitted bits in error.

The number of stuffing bits is a parameter peculiar to the READ Algorithm and therefore is included in Table 6-2 only. Likewise, only the 3M code employs an adaptive K-factor, and consequently, the number of lines encoded by the two dimensional algorithm is included in Table 6-3

* High Resolution - 2,376 lines x 1728 pels/lie = 4,105,728 pels
Standard Resolution - 1,188 lines x 1728 pels/line = 2,052,864 pels

TABLE 6-2 TEST RESULTS, READ ALGORITHM

TEST RUN	# CODED BITS	# BITS IN ERROR XMTD	BER $X10^{-3}$	# INCORRECT PELS	# STUFFING BITS	# CODED DATABITS	ESF	CF_3	CF_4
1	442,434	362	.82	21,030	7,497	390,927	58.093	4.6399	5.2513
2	727,418	564	.775	38,283	9,842	620,671	67.877	5.6442	6.6150
3	442,434	510	1.15	21,175	7,497	390,927	41.519	4.6399	5.2513
4	727,418	510	.701	21,197	9,842	620,671	41.562	5.6442	6.6150
5	442,434	296	.669	14,305	7,497	390,927	48.327	4.6399	5.2513
6	727,418	682	.937	50,035	9,842	620,671	73.36	5.6442	6.6150
7	442,434	598	1.35	19,657	7,497	390,927	32.87	4.6399	5.2513
8	727,418	793	1.09	39,569	9,842	620,671	49.89	5.6442	6.6150
9	727,418	564	.775	50,559	9,842	620,671	89.64	5.6442	6.6150
10	727,418	564	.775	40,294	9,842	620,671	71.44	5.6442	6.6150
11	419,636	290	.691	13,299	7,497	390,927	45.85	4.8920	5.2513
12	678,257	510	.751	31,561	9,842	620,671	61.88	6.0534	6.6150
13	419,636	290	.691	13,105	7,497	390,927	45.18	4.8920	5.2513
14	678,257	783	1.154	34,636	9,842	620,671	44.23	6.0534	6.6150
15	188,070	120	.638	3,538	1,654	113,956	29.48	10.915	18.0145
16	250,379	216	.862	8,591	2,171	174,838	39.77	16.398	23.4830
17	370,448	220	.593	19,920	3,623	322,307	90.545	11.083	12.738
18	253,989	216	.850	7,549	2,467	210,040	34.94	8.082	9.7737
19	423,040	290	.685	9,361	8,567	385,871	32.27	4.852	5.320
20	664,857	564	,848	25,657	12,503	616,812	45.49	6.175	6.656

TABLE 6-3 TEST RESULTS, 3M ALGORITHM

TEST RUN	# CODED BITS	# BITS IN ERROR IMTD	BER X10^{-3}	NUMBER INCORRECT PELS	NUMBER 2-DIM. LINES	NUMBER CODED DATABITS	ESF	CF_3	CF_4
1	441,104	362	.82	12,255	202	397,549	33.8536	4.6539	5.1638
2	757,869	564	.74	38,682	924	668,555	68.5851	5.4175	6.1412
3	441,104	510	1.15	16,253	202	397,549	31.8686	4.6539	5.1638
4	757,869	510	.67	19,435	924	668,555	38.1098	5.4175	6.1412
5	441,104	296	.67	11,977	202	397,549	40.4628	4.6539	5.1638
6	757,869	682	.89	47,401	924	668,555	69.5029	5.4175	6.1412
7	441,104	598	1.35	11,767	202	397,549	19.6772	4.6539	5.1638
8	757,869	793	1.04	31,900	924	668,555	40.2270	5.4175	6.1412
9	757,869	564	.74	32,650	924	668,555	57.8901	5.4175	6.1412
10	757,869	564	.74	34,597	924	668,555	61.3422	5.4175	6.1412
11	419,039	290	.69	12,538	250	396,899	43.2345	4.8990	5.1723
12	709,588	510	.71	29,229	1068	664,381	57.3118	5.7861	6.1798
13	419,039	290	.69	9,599	250	396,899	33.1000	4.8990	5.1723
14	709,588	793	1.11	25,898	1068	664,381	32.6582	5.7861	6.1798
15	192,484	132	.68	1,160	128	126,122	8.7879	10.6651	16.2768
16	260,382	216	.82	13,650	675	201,902	63.1944	15.7681	20.3353
17	392,062	220	.56	17,432	1446	346,058	79.2364	10.4721	11.8643
18	264,163	216	.81	7,386	375	226,815	34.1944	7.7712	9.0508
19	431,481	356	.82	8,485	262	399,497	23.8343	4.7577	5.1386
20	716,340	564	.78	21,333	1309	676,844	36.0514	5.7315	6.0660

TABLE 6-4 TEST RESULTS, IBM ALGORITHM

TEST RUN	# CODED BITS	# BITS IN ERROR XMTD	BER $X10^{-3}$	# INCORRECT PELS	# OF CODED DATABITS	ESF	CF_3	CF_4
1	430,215	346	.80	16,013	383,562	46.2803	4.7717	5.3521
2	727,740	564	.77	30,600	627,122	54.2553	5.6418	6.5469
3	430,215	510	1.18	15,842	383,562	31.0627	4.7717	5.3521
4	727,740	510	.70	18,102	627,122	35.4941	5.6418	6.5469
5	430,215	296	.62	9,004	383,562	30.4189	4.7717	5.3521
6	727,740	682	.93	43,186	627,122	63.3226	5.6418	6.5469
7	430,215	524	1.21	13,925	383,562	26.5744	4.7717	5.3521
8	727,740	793	1.08	24,117	627,122	30.4124	5.6418	6.5469
9	727,740	564	.77	33,899	627,122	60.1046	5.6418	6.5469
10	727,740	564	.77	37,025	627,122	65.6472	5.6418	6.5469
11	407,850	262	.64	11,329	383,562	43.2404	5.0334	5.3521
12	679,677	510	.75	25,028	627,122	49.0745	6.0407	6.5469
13	407,850	278	.68	13,920	383,562	50.0719	5.0334	5.3521
14	679,677	789	1.16	30,120	627,122	38.1749	6.0407	6.5469
15	187,619	120	.63	3,034	115,011	25.2833	10.9417	17.8493
16	255,119	216	.84	10,928	181,740	50.5926	16.0934	22.5912
17	37,810	220	.58	18,908	329,697	85.9454	10.8385	12.4530
18	254,459	216	.84	8,211	210,809	38.0139	8.0676	9.7380
19	413,042	272	.65	6,056	379,460	22.2647	4.9701	5.4100
20	672,892	564	.83	27,093	628,606	48.0372	6.1016	6.5315

TABLE 6 5 TEST RESULTS. XEROX ALGORITHM

TEST RUN	# CODED BITS	# BITS IN ERROR XMTD	BER $X10^{-3}$	# INCORRECT PELS	# OF CODED DATA BITS	ESF	CF_3	CF_4
1	468,341	374	.798	15,642	430,660	41.8235	4.3633	4.7668
2	822,790	564	.685	25,464	748,406	45.1489	4.9900	5.4860
3	468,341	510	1.089	12,336	430,660	24.1882	4.3833	4.7668
4	822,790	510	.620	14,789	748,406	28.9980	4.9900	5.4860
5	468,341	326	.696	10,753	430,660	32.9846	4.3833	4.7668
6	822,790	682	.829	29,913	748,406	43.8607	4.9900	5.4860
7	468,341	626	1.336	11,229	430,660	17.9377	4.3833	4.7668
8	822,790	899	1.093	19,133	748,406	21.2825	4.9900	5.4860
9	822,790	564	.685	25,984	748,406	46.0709	4.9900	5.4860
10	822,790	564	.685	32,316	748,406	57.2979	4.9900	5.4860
11	448,833	362	.806	14,087	430,660	38.9144	4.5738	4.7668
12	782,611	510	.652	13,119	748,406	25.7235	5.2462	5.4860
13	448,833	296	.659	11,444	430,660	38.6622	4.5738	4.7668
14	782,611	793	1.013	15,660	748,406	19.7478	5.2462	5.4860
15	198,749	132	.664	2,571	133,050	19.4773	10.3289	15.4293
16	290,172	220	.758	9,556	233,764	43.4364	14.1493	17.5636
17	447,691	362	.809	13,802	419,000	38.1271	9.1709	9.7989
18	269,544	220	.816	3,041	236,284	13.8227	7.5161	8.6881
19	448,809	362	.807	9,017	421,857	24.9088	4.5740	4.8663
20	779,185	564	.724	18,894	749,859	33.5000	5.2693	5.4753

TABLE 6-6 TEST RESULTS, AT&T ALGORITHM

TEST RUN	NO. CODED BITS	NO. BITS IN ERROR XMTD	BER $\times 10^{-3}$	NO. INCORRECT PELS	NO. OF CODED DATA BITS	ESF	CF_3	CF_4
1	466,613	374	.80	19,378	415,034	51.8128	4.3995	4.9463
2	763,481	564	.73	33,756	655,807	59.8511	5.3776	6.2606
3	466,613	510	1.09	13,823	415,034	37.1039	4.3995	4.9463
4	763,481	510	.66	22,773	655,807	44.6529	5.3776	6.2606
5	466,613	326	.69	15,143	415,034	46.4509	4.3995	4.9463
6	763,481	682	.89	40,440	655,807	59.2962	5.3776	6.2606
7	466,613	626	1.34	11,529	415,034	18.4169	4.3995	4.9463
8	763,481	793	1.03	24,628	655,807	31.056	5.3776	6.2606
9	763,481	564	.73	34,127	655,807	60.5089	5.3776	6.2606
10	763,481	564	.73	36,800	655,807	65.2482	5.3776	6.2606
11	443,326	362	.81	15,862	415,034	43.8177	4.6306	4.9463
12	714,016	510	.71	27,731	655,807	54.3745	5.7502	6.2606
13	443,326	296	.66	10,432	414,034	35.2432	4.6306	4.9463
14	714,016	793	1.11	24,958	655,807	31.4729	5.7502	6.2606
15	193,573	132	.68	1,236	112,546	9.3636	10.6051	18.2402
16	258,832	216	.83	8,586	175,159	39.7500	15.8625	23.4400
17	388,419	220	.56	15,238	335,235	69.2636	10.5704	12.2473
18	267,503	220	.82	5,570	220,429	25.3182	7.6742	9.3130
19	451,171	362	.80	9,463	415,929	26.1409	4.5501	4.9356
20	709,814	564	.79	25,074	663,918	44.4574	5.7842	6.1841

only.

The CCITT suggested that experimenters should measure the statistics related to the number of bits required to define the individual scan lines. Statistics which were measured are minimum bits/line, maximum bits/line, average bits/line, and standard deviation. Of the twenty test runs, only 10 give independent data related to coded line statistics. The parameters of these ten independent measurements are tabulated in Table 6-7 along with the test results for the READ code. Table 6-8 gives the results for the 3M and IBM codes, while Table 6-9 tabulates the data for the Xerox and AT&T algorithms.

6.2 Summary of Compression Data

Two types of compression data were computed in the simulation process - transmitted bits and compression factor. Since these parameters are merely two different measures of the same function, it was decided, for reasons of simplicity, to summarize the compression data using only the transmitted bits. The number of coded data bits are summarized for all algorithms in Table 6-10 while the total bits including overhead is tabulated in Table 6-11.

At the bottom of each table, the average number of bits for high resolution and low resolution tests is computed. It is recognized that such an averaging process infers an equal weighting of the component test runs. It is also recognized that this is probably not the optimum weighting function to be applied to the data. Nevertheless, the average was computed primarily as a data reduction process and can

TABLE 6-7 CODED LINE LENGTH STATISTICS - READ ALGORITHM

TEST PARAMETERS				TEST RESULTS - READ			
TEST DESIGNATION	TEST DOCUMENT #	MIN. SCAN. LINE TIME (ms.)	VERTICAL RESOLUTION	MIN. BITS/LINE	MAX. BITS/LINE	AVG. BITS/LINE	STANDARD DEVIATION
A	4	20	3.85	96	1,208	366.12	332.06
B	4	20	7.70	96	1,088	302.05	268.15
C	4	10	3.85	48	1,208	346.89	348.12
D	4	10	7.70	48	1,088	281.31	284.59
E	1	20	3.85	96	792	156.94	145.94
F	1	10	7.70	48	792	104.46	130.04
G	5	10	7.70	48	1,058	154.38	159.23
H	5	20	3.85	96	1,058	211.76	181.24
I	7	20	3.85	96	713	348.87	175.49
J	7	10	7.70	48	713	274.54	163.15

TABLE 6-8 CODED LINE LENGTH STATISTICS - 3M, IBM ALGORITHMS

ALGORITHM	TEST DESIG-NATION	MINIMUM BITS/LINE	MAXIMUM BITS/LINE	AVERAGE BITS/LINE	STANDARD DEVIATION
3M	A	96	1,089	371.23	337.14
	B	96	1,058	318.94	287.90
	C	48	1,089	352.66	352.72
	D	48	1,058	298.62	303.58
	E	96	813	161.96	154.18
	F	48	792	109.56	137.87
	G	48	1,055	164.98	167.83
	H	96	1,063	222.29	191.44
	I	96	718	363.13	184.50
	J	48	707	301.46	173.05
IBM	A	96	1,104	362.07	329.55
	B	96	1,089	306.25	273.04
	C	48	1,104	343.24	345.17
	D	48	1,089	286.03	289.12
	E	96	797	157.86	146.30
	F	48	797	107.34	132.95
	G	48	1,063	159.40	161.22
	H	96	1,063	214.13	182.45
	I	96	718	347.61	175.64
	J	48	718	283.17	165.40

TABLE 6-9 CODED LINE LENGTH STATISTICS - XEROX, ATT ALGORITHMS

ALGORITHM	TEST DESIGNATION	MINIMUM BITS/LINE	MAXIMUM BITS/LINE	AVERAGE BITS/LINE	STANDARD DEVIATION
XEROX	A	96	1,240	395.11	355.07
	B	96	1,091	347.81	303.63
	C	48	1,240	378.11	369.79
	D	48	1,091	330.32	318.59
	E	96	927	167.40	161.42
	F	48	831	122.53	152.90
	G	48	1,062	188.67	174.59
	H	96	1,062	228.08	191.46
	I	96	717	378.36	183.17
	J	48	931	328.04	180.43
AT&T	A	96	1,357	392.71	375.48
	B	96	1,357	321.3	305.02
	C	48	1,357	373.11	391.36
	D	48	1,357	300.48	320.88
	E	96	938	162.88	163.02
	F	48	938	108.91	143.43
	G	48	1,269	-	-
	H	96	1,269	225.11	207.72
	I	96	882	379,71	207.70
	J	48	882		

TABLE 6-[illegible] SUMMARY OF CODED DATA BITS

TEST DOCUMENT #	VERT. RESOL. lines/mm.	JAPAN	3M	IBM	XEROX	AT&T
4	3.85	390,927	397,549	383,562	430,660	415,034
4	7.7	620,671	668,555	627,122	748,406	655,807
1	3.85	113,956	126,122	115,011	133,050	112,546
1	7.7	174,838	201,902	181,740	233,764	175,159
5	3.85	210,040	226,815	210,809	236,284	220,429
5	7.7	322,307	346,058	329,697	419,000	335,235
7	3.85	385,871	399,497	379,460	421,857	415,929
7	7.7	616,812	676,844	628,606	749,859	663,918
AVG.	3.85	275,198	287,497	272,210	305,463	290,984
AVG.	7.7	433,657	473,340	441,791	537,757	457,530
OVERALL AVG.		354,427	380,418	357,000	421,610	374,257

TABLE 6-11 SUMMARY OF CODED BITS

TEST DOC. #	MIN. SCAN LINE TIME (ms.)	VERT. RESOL. lines/mm.	JAPAN	3M	IBM	XEROX	AT&T
4	10	3.85	419,636	419,039	407,850	448,833	443,326
4	10	7.7	678,257	709,588	679,677	782,611	714,016
4	20	3.85	442,434	441,104	430,215	468,341	466,613
4	20	7.7	727,418	757,869	727,740	822,790	763,481
1	20	3.85	188,070	192,484	187,619	198,749	193,573
1	10	7.7	250,379	260,382	255,119	290,172	258,832
5	20	3.85	253,989	264,163	254,459	269,544	267,503
5	10	7.7	370,448	392,062	378,810	447,691	388,419
7	20	3.85	423,040	431,481	413,042	448,809	451,171
7	10	7.7	664,857	716,340	672,892	779,185	709,814
AVG.		3.85	345,434	349,654	338,637	366,855	364,437
AVG.		7.7	538,272	567,248	542,847	624,490	566,912
OVERALL AVERAGE			441,853	458,451	440,742	495,672	465,674

be considered as one possible basis for weighting the data. For similar reasons, the overall average of all runs, combining both high and low resolution, has been computed.

6.3 Summary of Error Sensitivity Data

All of the 20 computer runs provide a separate and distinct error sensitivity measurement. In an effort to reduce the data, the average ESF for all 20 runs has been computed for each algorithm and tabulated below.

Algorithm	$ESF_{avg.}$ for 20 runs
Japan	52.2
3M	43.7
IBM	44.7
Xerox	32.8
AT&T	42.2

It is recognized that this averaging process heavily weights test document number 4, error phase zero, transmission error file one, and an MSLT of 20 ms. Nevertheless, this averaging process would appear to be one meaningful way to weigh the data and reduce it.

7.0 REFERENCES

1. CCITT Contribution No. 66, "Criteria for the Evaluation of Two-Dimensional Coding Techniques for use in Digital Facsimile Terminals" Source: United States of America; Date: January 1979.
2. CCITT Contribution COM XIV - No. 70, "Report of the Meeting Held in Geneva," 11-15 Dec. 1978, Annex No. 2, Section III.
3. National Communications System Report, "Development of a Computer Program for Measuring the Compression and Error Sensitivity of Facsimile Coding Techniques," August 10, 1979.
4. CCITT Contribution COM XIV - No. 77-E, British Algorithm.
5. CCITT Contribution COM XIV - No. 82-E, Federal Republic of Germany Algorithm.
6. National Communications System Report, "Measurement of Compression Factor and Error Sensitivity Factor of German and British Two-Dimensional Facsimile Coding Techniques," anticipated publication date - October 1979.
7. Collective Letter No. 87 from the CCITT to Members of Study Group XIV COM/TO dated 21 May 1979, page 5, section 4.0.
8. Federal Republic of Germany, "Sensibility of Redundancy Reducing Codes to Transmission Bit Errors," CCITT Study Group XIV - Contribution No. 5, February 1977.

APPENDIX A

CCITT STUDY GROUP XIV

CONTRIBUTION No. 42

Source: Japan

International Telegraph and Telephone
Consultative Committee
(CCITT)

Period 1977-1980

COM XIV-No. 42-E

Original : English

Question : 2/XIV

Date : 28 August 1978

STUDY GROUP XIV - CONTRIBUTION No. 42

SOURCE : JAPAN

TITLE : PROPOSAL FOR DRAFT RECOMMENDATION OF TWO-DIMENSIONAL CODING SCHEME

1. Introduction

In order to meet the increasing demand for high-speed document facsimile over telephone-type circuits, CCITT Study Group XIV has been conducting study on the standardization of Group 3 machines. At the meeting of November 1977, a draft Recommendation T.4 related to Group 3 machines was produced.

In Paragraph 4.1, Section 4 "Coding Scheme" of the draft Recommendation T.4, the one-dimensional coding scheme using the Modified-Huffman code is defined. In Paragraph 4.2, it is noted that the one-dimensional coding scheme may be extended as an option to a two-dimensional scheme, and this is the subject of further study.

Since there have been strong demands for higher speed transmission of documents, the facsimile equipments which adopt the two-dimensional coding schemes having higher compression factors, compared with the one-dimensional coding scheme are nowadays widely used in Japan and in several other countries.

This fact proves the superiority of the two-dimensional coding schemes, which enable a remarkable speeding-up of document transmission with scarcely any increase in system cost and without any great degradation in quality of received copies caused by transmission errors, compared with the one-dimensional coding scheme.

Japan has previously pointed out the effectiveness of two-dimensional coding schemes, and has several times presented contributions showing actual coding schemes and comparative data on compression factors at the Group 3 Special Rapporteur's Meetings of the last study period. Furthermore, at the Study Group Meeting of November 1977, Japan presented a contribution (COM XIV-No. 15-E) stating that a two-dimensional coding scheme should be selected from among two-dimensional line-by-line coding schemes, and was requested to present a further detailed contribution on this subject.

Under these circumstances, Japan has earnestly studied two-dimensional coding schemes in order to make possible the further speeding-up of Group 3 machines. As a result of these studies, in order to complete the draft Recommendation T.4, Japan proposes the following system as the two-dimensional coding scheme for the draft Recommendation.

2. Coding Scheme

The coding scheme proposed is a two-dimensional coding scheme, called READ (Relative Element Address Designate) coding, which can be easily extended from the one-dimensional coding scheme.

Coding algorithm, line-synchronization signal, FILL, setting of Parameter K and RTC signal are described in detail in ANNEX 1, in a format capable of being inserted in Paragraph 4.2 of the draft Recommendation T.4.

3. Coding Efficiency

The transmission times obtained by computer simulation are shown in ANNEX 2. The eight documents contained in magnetic tapes offered by French PTT were used as the test documents.

Compared with the one-dimensional coding scheme, the ratio of transmission time is 82.7 % (K = 2) for the normal definition standard, and 62.9 % (K = 4) for the higher definition standard.

When transmission errors can be corrected by retransmission, such as in data networks, Parameter K can be set to $K = \infty$. In this case, the ratio of transmission time will become 65.5 % for the normal definition standard and 50.7 % for the higher definition standard, compared with the one-dimensional coding scheme.

4. Conclusion

The proposed coding scheme, at the transmission rate of 4800 bps, is capable of transmitting the test documents in an average of 47 seconds for the normal definition standard, and in an average of 73 seconds for the higher definition standard.

The two-dimensional coding scheme proposed here is superior in respect to compression factor compared with conventional coding schemes, while possessing ample capability against error vulnerability and simplicity in system configuration.

For the document facsimile of complicated ideographic characters (Chinese characters, for example) in wide-spread use over the world, transmission employing higher definition standard is indispensable. In such cases, the effectiveness of this coding scheme, which can greatly shorten the transmission time in particular, is extremely great.

In addition, while this coding scheme can be easily made to interwork with the standard one-dimensional system, in the case of data networks with superior line quality that have adopted error correction, this coding scheme can be applied with Parameter K set to $K = \infty$. This will result in the further speeding up of transmission.

In view of the features described above, Japan proposes that the READ coding scheme should be adopted as the two-dimensional system for Group 3 machines.

Annexes : 2

ANNEX 1

4.2 Two-dimensional coding scheme

The two-dimensional coding scheme shall be used as an extension of the one-dimensional coding scheme specified in Paragraph 4.1.

a) DATA

(1) Parameter K

In order to limit the disturbed area in the event of transmission errors, one-dimensional coding is applied for each first line of every successive K lines and two-dimensional coding is applied for the following (K-1) lines. The value of K is called Parameter K.

(2) One-dimensional coding

This conforms with the description of Item a) DATA, Paragraph 4.1.

(3) Two-dimensional coding

1) Starting picture element and changing picture element

A changing picture element is the colour changed picture element from black to white, or vice versa, along a scanning line.

In two-dimensional coding, each coding line is coded line-by-line, using the preceding changing picture element along the coding line or the changing picture element on the reference line which is just above the coding line. The coding line which has been coded becomes the new reference line for the next coding line.

The starting picture element and the changing picture elements shall be defined as follows: (see Figure 3)

a_0: Starting picture element on the coding line which becomes the reference picture element

a_1: Changing picture element on the coding line, next to a_0

a_2: Changing picture element on the coding line, next to a_1

b_1: First changing picture element, whose colour is opposite to a_0, which occurs on the reference line after the picture element just upon a_0

b_2: Changing picture element coming after b_1 on the reference line

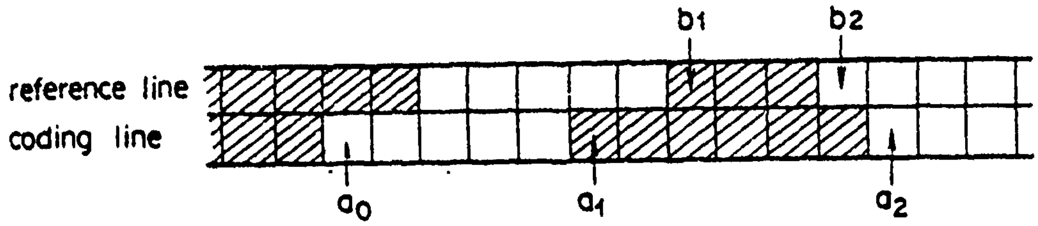


Figure 3 - Starting picture element and
changing picture elements

2) Coding modes

The following three kinds of coding modes are adopted and selectively used, according to the procedure described in Paragraph 2.3.

(i) Pass mode

As shown in Figure 4, the state where changing picture elements b_1 and b_2 on the reference line are detected before changing picture element a_1 is defined as Pass mode.

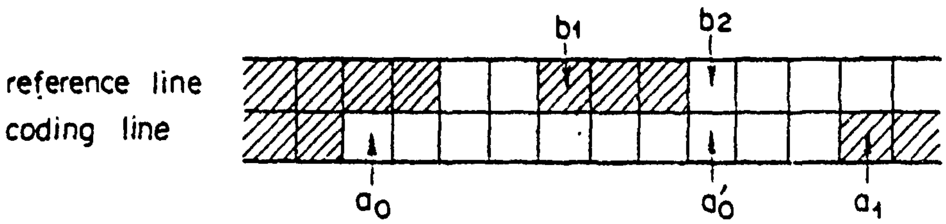


Figure 4 - Pass mode

However, the state where b_2 occurs just upon a_1, as shown in Figure 5, is not considered as Pass mode.

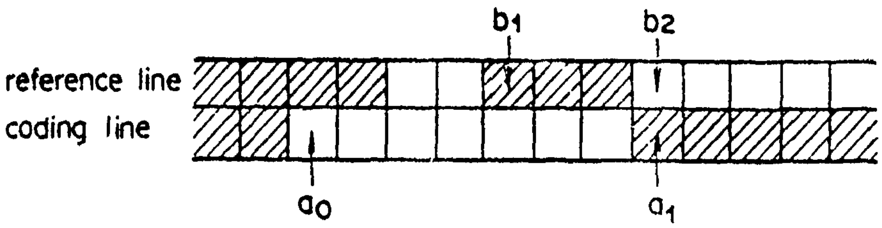


Figure 5 - An example not corresponding to Pass mode

(ii) Horizontal mode

The coding of distance a_0a_1 and distance a_1a_2 on the coding line is defined as Horizontal mode. (see Figure 6)

(iii) Vertical mode

The coding of the relative distance a_1b_1 between the changing picture element b_1 on the reference line and the changing picture element a_1 on the coding line is defined as Vertical mode. (see Figure 6)

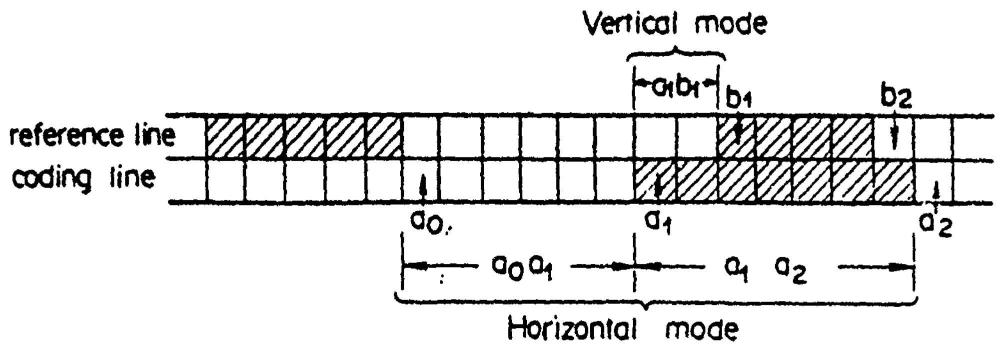


Figure 6 - Vertical mode and Horizontal mode

3) Coding procedure

The coding line and the reference line are observed at the same time from left to right, and the changing picture elements are detected successively. Then, whenever one of Pass, Vertical or Horizontal modes is detected, the code shown in Table 3 is generated.

Step 1

When Pass mode is detected, it is coded as Pass mode code "1110", as shown in Table 3. After this processing, picture element a_0' just under b_2 is regarded as the new starting picture element a_0 for the next coding. (see Figure 4)

Step 2

When a_1 is detected on the coding line before b_2 is detected on the reference line, adaptive coding is performed. That is, as shown in Table 3, the number of bits $\{a_0a_1\}$ (the sum of the 4 bits of Horizontal mode code "1111" and the number of bits obtained by one-dimensional coding of a_0a_1) and the number of bits $\{a_1b_1\}$ obtained by Vertical mode coding of a_1b_1, are measured respectively and the number of bits of both codings are compared. (see Figure 6)

(i) Case 1: $(a_0a_1) > (a_1b_1)$

As shown in Table 3, a_1b_1 is coded by Vertical mode, after which position a_1 is regarded as new starting picture element a_0 for the next coding.

(ii) Case 2: $(a_0a_1) \leq (a_1b_1)$

Scanning is performed until changing picture element a_2 on the coding line is detected. Then, as shown in Table 3, following Horizontal mode code "1111", a_0a_1 and a_1a_2 are respectively coded by one-dimensional coding. After this processing position a_2 is regarded as the new starting picture element a_0 for the next coding.

4) <u>Processing first and last picture elements in a line</u>

(i) Processing first picture element

The first starting picture element a_0 on each coding line is imaginarily set at a position just before the first picture element, and is regarded as a white picture element. In this case, the distance a_0a_1 is replaced by a_0a_1-1.

(ii) Processing last picture element

Coding of each line is ended after coding of the changing picture element a_1 or a_2 positioned imaginarily next to the last picture element. If b_1 or b_2 is not detected on the reference line, an imaginary picture element just after the last picture element is assumed as b_1 or b_2. A flow diagram for this coding scheme is shown in APPENDIX 1, and coding examples in APPENDIX 2.

TABLE 3

Code table

Mode	Elements to be coded		Notation	Code
Pass mode	b_1 , b_2		P	1110
Horizontal mode	a_0a_1 , a_1a_2		$H(a_0a_1, a_1a_2)$	$1111 + M(a_0a_1) + M(a_1a_2)$
Vertical mode	a_1 just under b_1	$a_1b_1 = 0$	$V(0)$	0
	a_1 on the right of b_1	$a_1b_1 = 1$	$V_R(a_1b_1)$	100
		$a_1b_1 \geq 2$		$1100 + D(a_1b_1 - 1)$
	a_1 on the left of b_1	$a_1b_1 = 1$	$V_L(a_1b_1)$	101
		$a_1b_1 \geq 2$		$1101 + D(a_1b_1 - 1)$

(Note 1) Code D(n) represents the following n bit code:

$$D(n) : \underbrace{\underbrace{00\cdots\cdots\cdots0}_{(n-1)\ \text{bits}}1}_{n\ \text{bits}}$$

(Note 2) Code M( ) of Horizontal mode represents the code words in Tables 1 & 2, Paragraph 4.1, the draft Recommendation T.4.

b) Line synchronization signal

One of the two line synchronization signals, LSS1 or LSS2, precedes the data signals of the one-dimensional or two-dimensional coding lines respectively.

Format:

(i) The line synchronization signal to be added to the head of the one-dimensional coding line

LSS1 : 01111111

(ii) The line synchronization signal to be added to the head of the two-dimensional coding line

LSS2 : 01111110

To make the signals LSS1 and LSS2 unique, a "0" is inserted in the data stream after any occurrence of five continuous "1"s.

c) FILL

FILL is inserted between a line of DATA and the line synchronization signal, LSS1 or LSS2, but is not inserted in DATA.

Format : variable length string of 0's.

d) Setup of Parameter K

Parameter K should be fixed as follows.

Normal definition standard : K = 2
Higher definition standard : K = 4

e) Return to control (RTC)

The end of a document transmission is indicated by sending six consecutive LSS1's. Following the RTC signal, the transmitter will send the post message command in the Recommendation T.30 framed format at the data rate.

Format : 01111111 01111111

(total of 6 times LSS1)

To further clarify the relationship of the signals defined herein, Figures 7 and 8 are offered in the case of K = 2. Figure 7 shows several scan lines of data starting at the beginning of a transmitted page. Figure 8 shows the last several lines of a page.

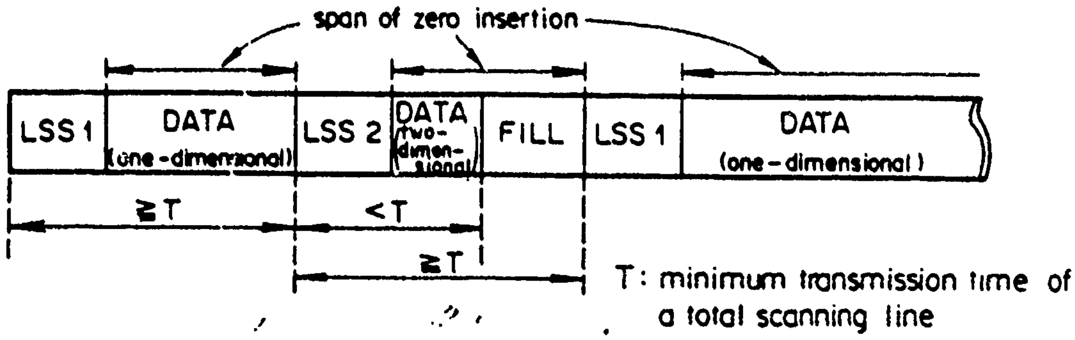


Figure 7 - Message transmission (first part of page)

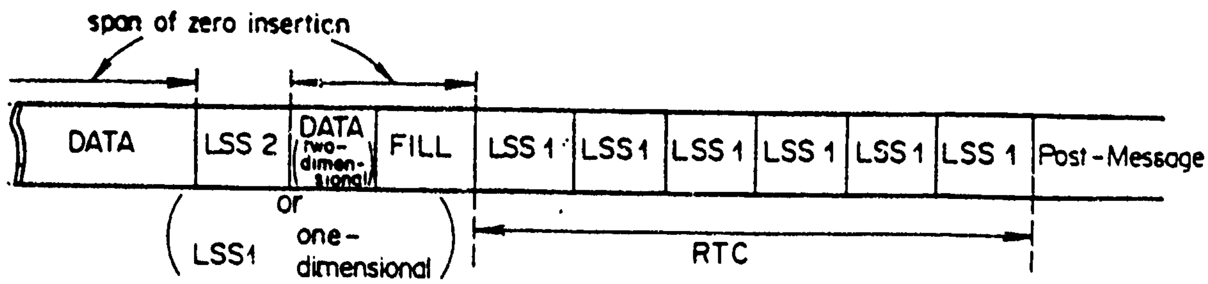
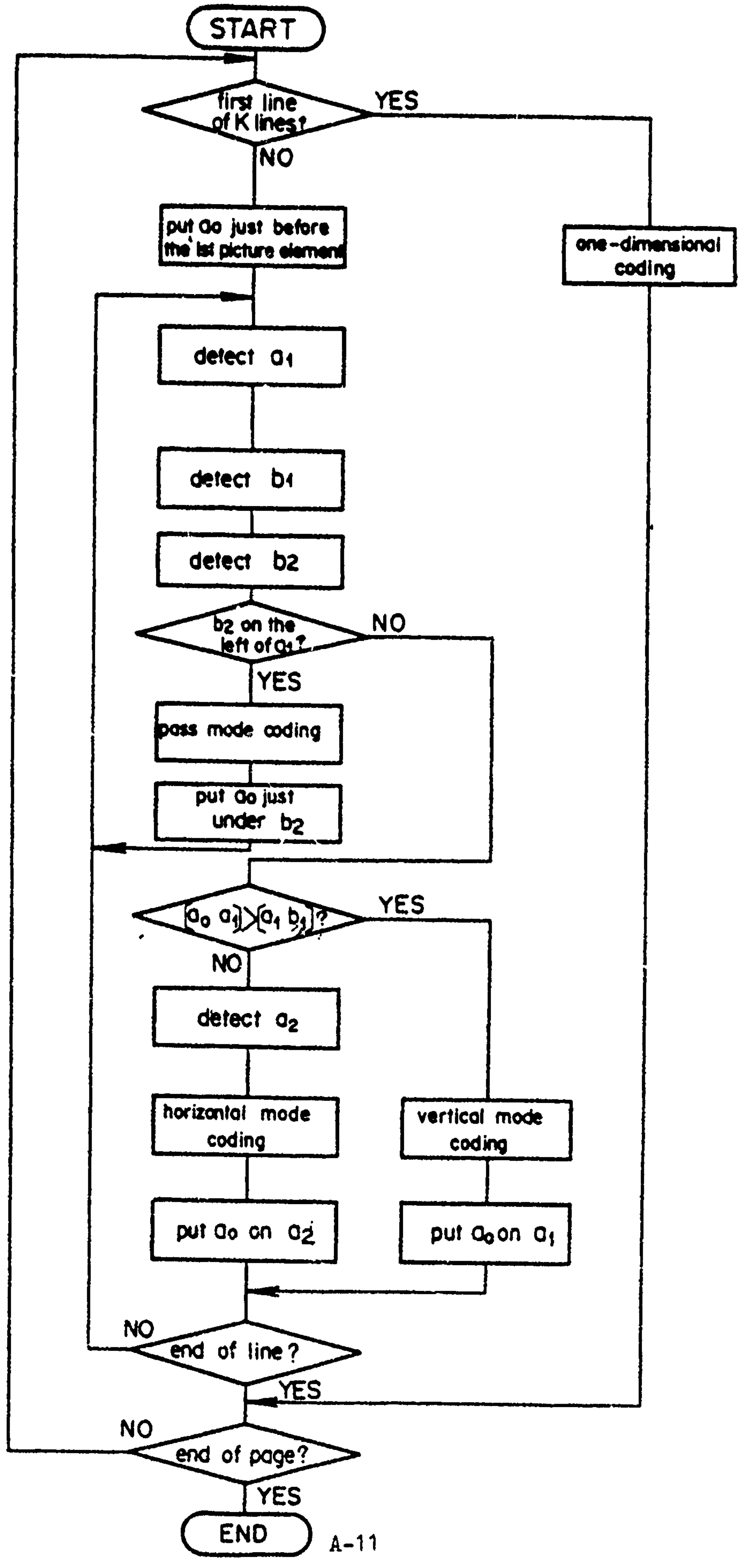


Figure 8 - Message transmission (last part of page)

Appendices : 2

Appendix 1
(to Annex 1)

Coding flow diagram



Appendix 2
(to Annex 1)

Coding examples

Figure (a) shows coding examples of the first part of scanning lines and Figure (b) coding examples of the last part, while Figure (c) shows other coding examples. The notations P, H and V in the figures are, as shown in Table 3, the symbols for Pass mode, Horizontal mode and Vertical mode respectively. The picture elements marked with black spot indicate the changing picture elements to be coded.

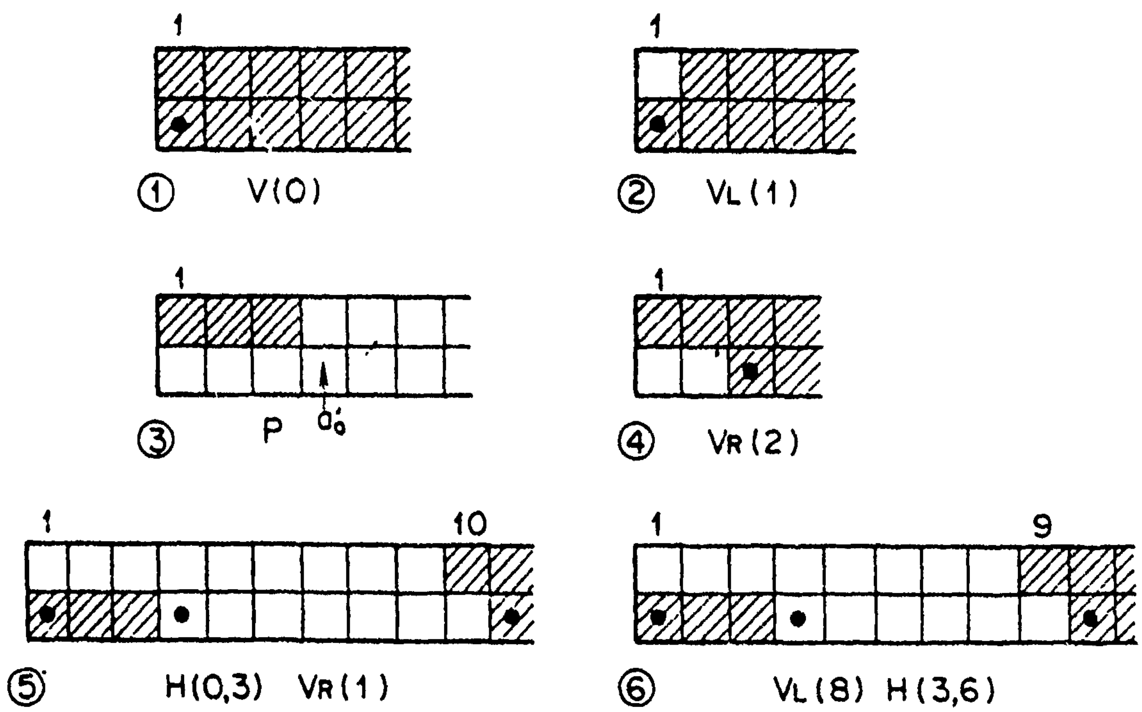


Figure (a) - Coding examples (a) : first part of scanning line

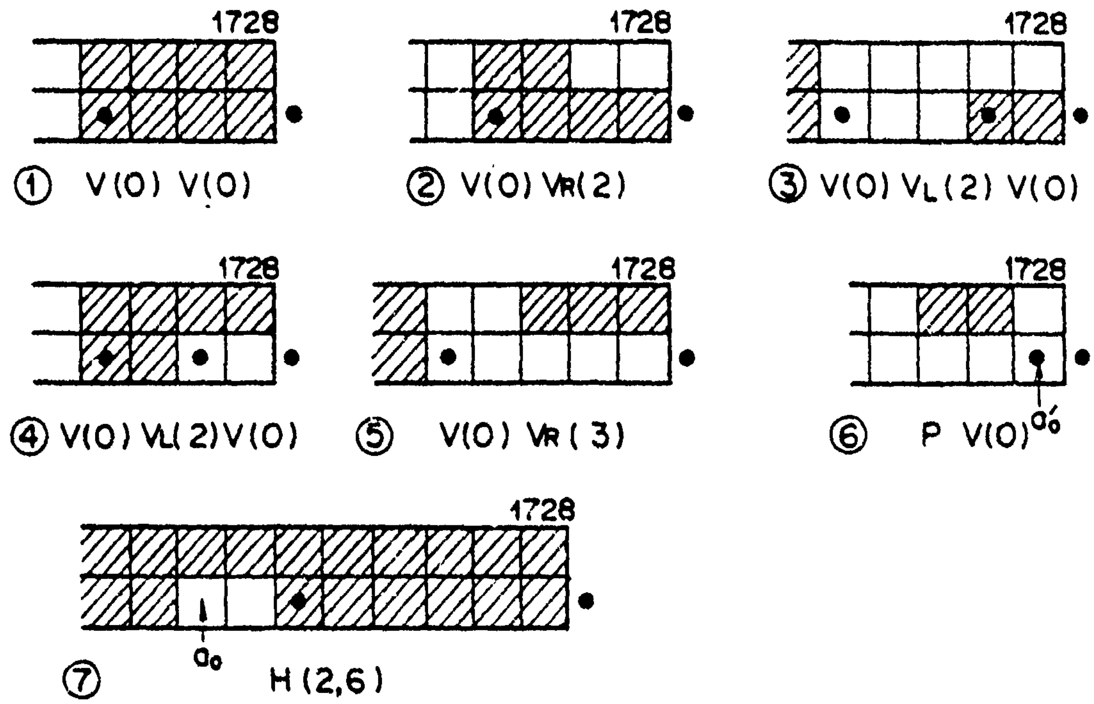


Figure (b) - Coding examples (b) : last part of scanning line

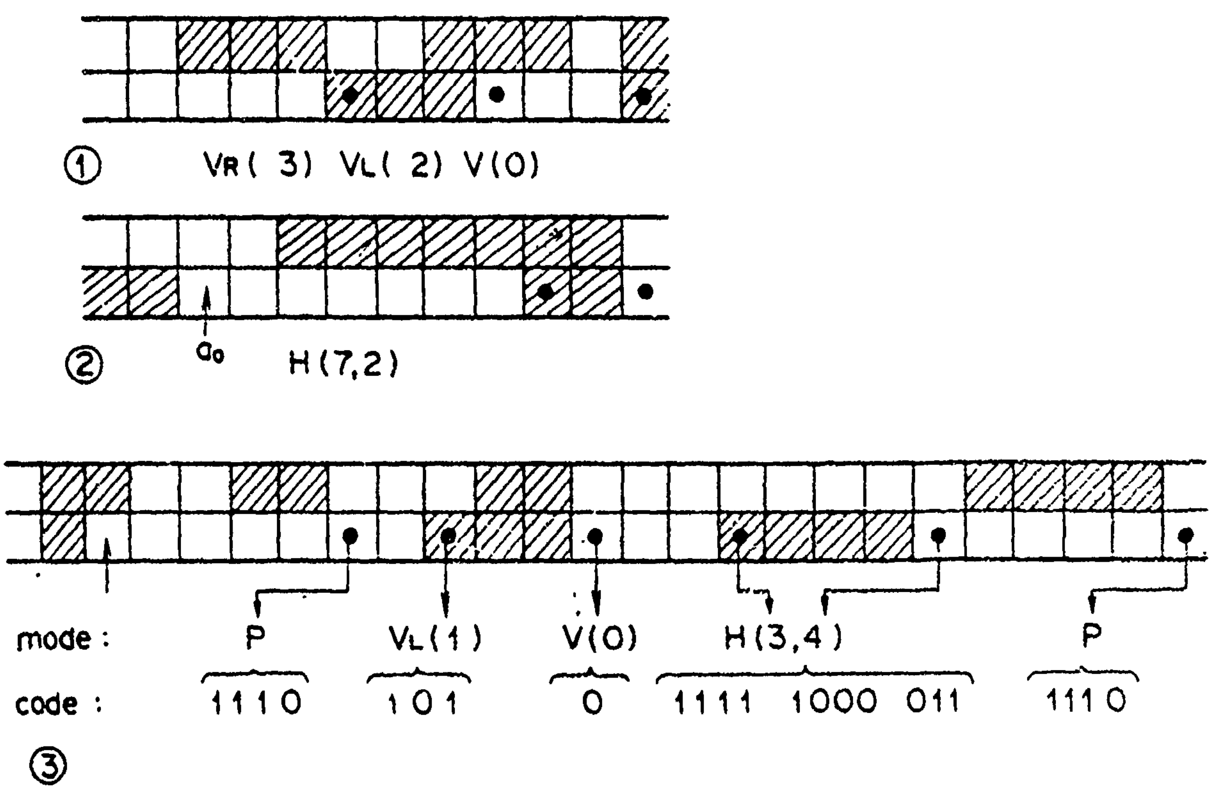


Figure (c) - Coding examples (c)

ANNEX 2-1

Table (a) – Average transmission time (seconds/page)

Coding method: Control code	Resolution	One-dimensional coding	Two-dimensional coding: K = 2	K = 4	K = ∞
with control code (minimum transmission time 5msec)	8 x 4	56.3	46.6	42.2	37.8
	8 x 8	112.5	85.2	72.6	60.0
without control code	8 x 4	53.3	44.1	39.5	34.9
	8 x 8	106.6	80.2	67.1	54.0

(Transmission rate : 4800 bps)

ANNEX 2-2

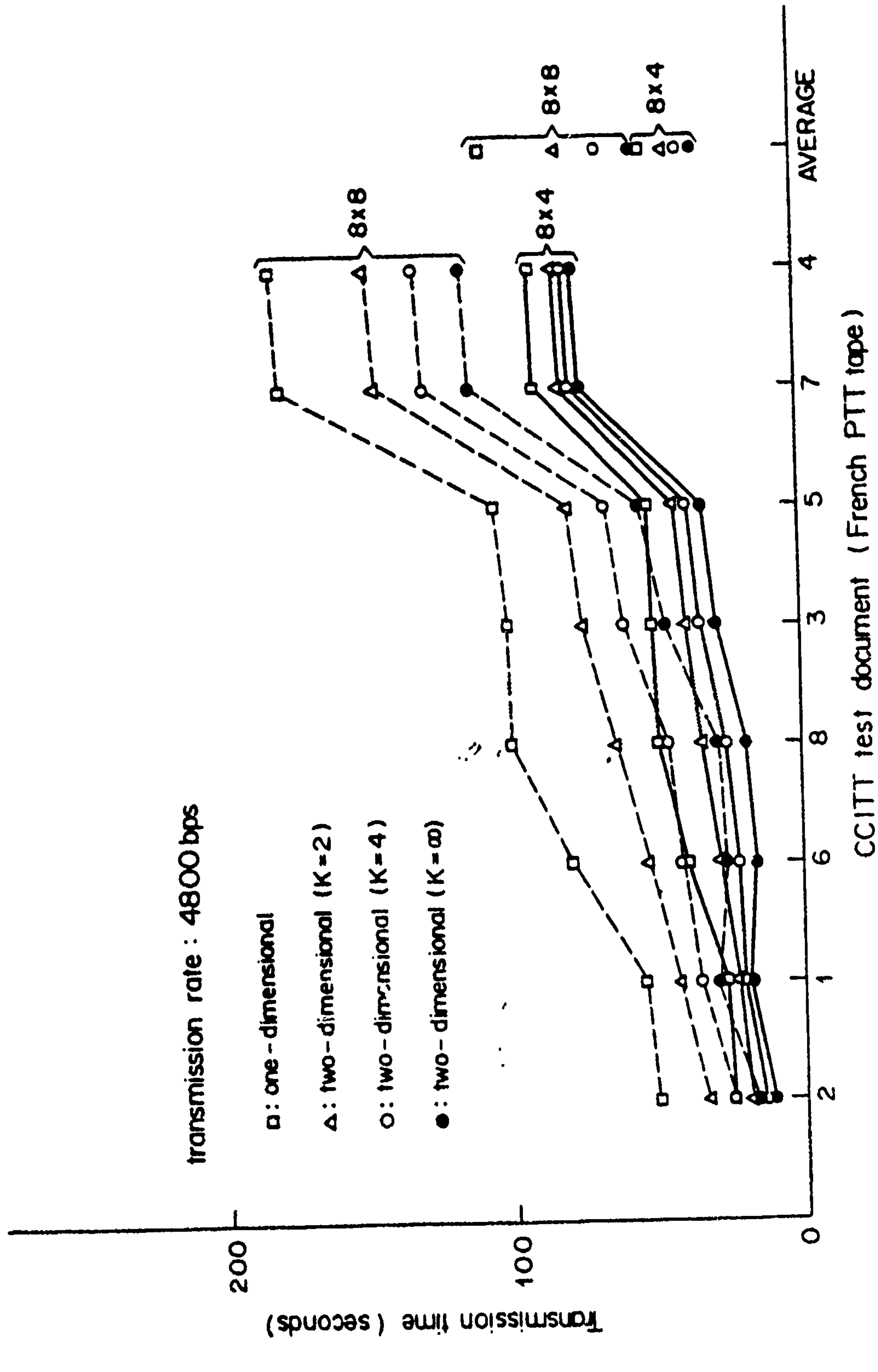


Figure (d) - Transmission time (without control code)

ANNEX 2-3

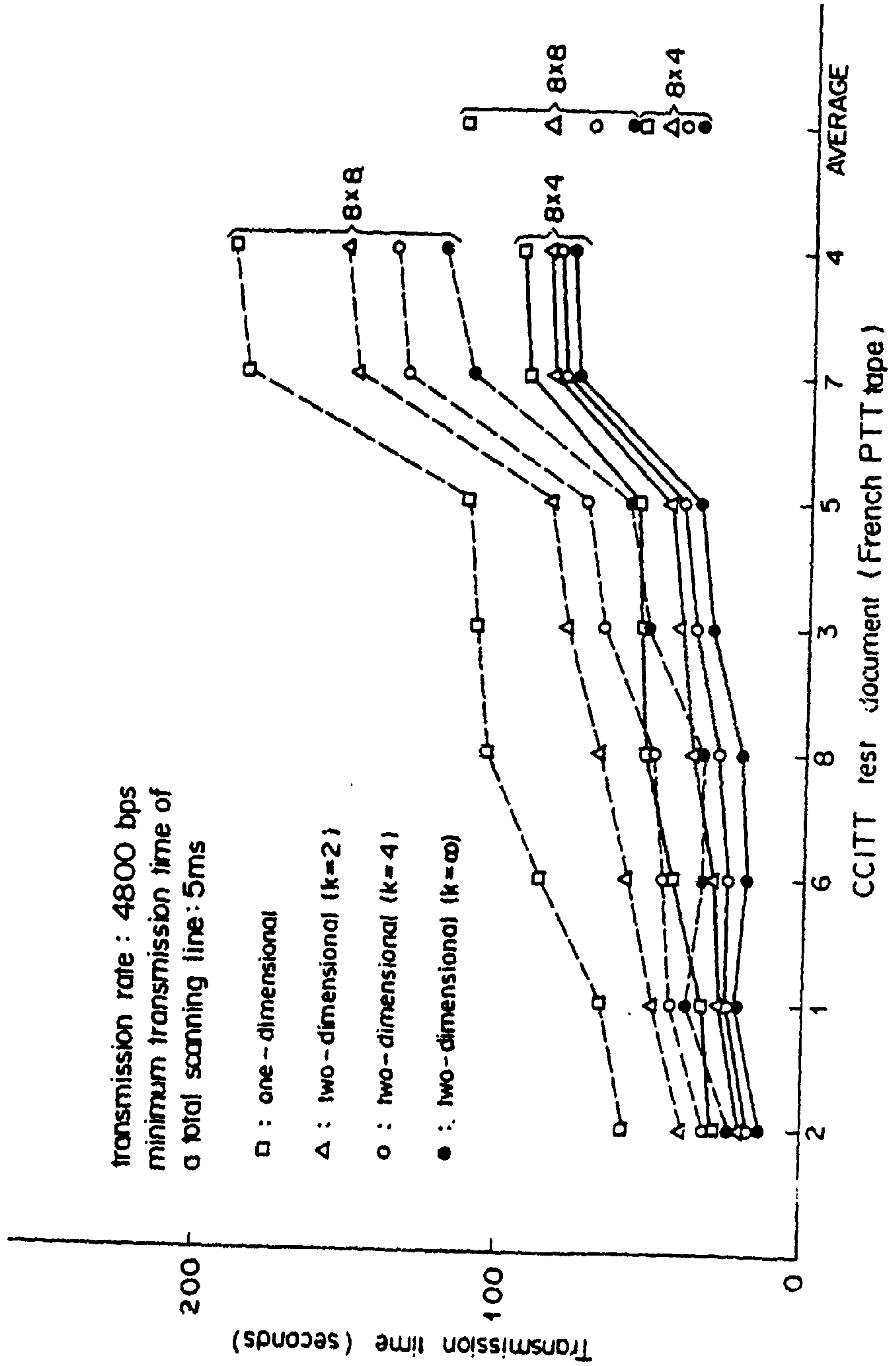


Figure (e) - Transmission time (with control code)

APPENDIX B

CCITT STUDY GROUP XIV

CONTRIBUTION No. 74

Source: 3M Company

International Telegraph and Telephone
Consultative Committee
(CCITT)

Period 1977-1980

COM XIV-No. 74-E

Original : English

Question : 2/XIV

Date : March 1979

STUDY GROUP XIV - CONTRIBUTION NO. 74

SOURCE : 3M COMPANY

TITLE : RESULTS OF STUDIES ON PREVIOUSLY PROPOSED TWO-DIMENSIONAL CODING SCHEMES

1.0 INTRODUCTION

3M Company has studied the EDIC code, the READ code, and the IBM code in an attempt to find a workable compromise. 3M realizes that such a compromise should be applicable not only to the authors of the prospective code, but to all the CCITT participants.

In attempting to find this compromise, the various compression algorithms were combined into one generalized model wherein each parameter was expressed in abstract terms rather than actual codes. The parameters were then individually analyzed and assigned code lengths using the concepts and statistical data disclosed in the prior compression schemes. The individual results of this parametric analysis were combined to generate a detailed code table and an encoding algorithm. During this process, ease of implementing the resultant algorithm was a constant consideration.

Since this encoding algorithm was generated to a large degree by studying other code sets, 3M claims no rights to the resultant code algorithm. Furthermore, the improvements/modifications to the code set are offered to aid in reaching a compromise on a two-dimensional algorithm. Therefore, keeping with

this spirit, no private claims are made to these modifications.

This paper assumes that the reader is familiar with the compression algorithms beinn discussed. If background information is needed, please refer to References 1, 2, 3, and 4.

2.0 CODE SELECTION FOR EACH PARAMETER

2.1 Line Synchronization Signal

The Line Synchronization Signal, LSS, serves two purposes. It specifies, non-ambiguously, the beginning and end of each compressed line, and it gives information on how to decode the ensuing line.

The choice for the LSS was eleven zeros, followed by a one, followed by a tag bit indicating the type of run to follow (i.e., "0" indicates one-dimensional codino and "1" indicates two-dimensional coding). The reasons for this choice are as follows.

i) Using an HDLC flag code is undesirable since it is a level 2 link control signal and should be kept independent from data compression algorithm which is on a higher level in the layered protocol structure (ref. Temporary Doc. No. 8-E CCITT SG XIV 12/11-12/15). Furthermore, use of the HDLC flag necessitates zero insertion.

ii) The mechanism is already in place to receive, with reasonable protection, the standard one-dimensional EOL code. This mechanism includes a method for handling fill data and doing a line by line check on the data (i.e., when EOL is detected, exactly 1728 pels would have been decoded).

2.2 Frequency of Resynchronization

The repetition period, expressed in number of lines, at which the scanned data should be compressed using the one-dimensional algorithm is the parameter K. It was decided that K should be adaptive but confined within the bounds of K less than or equal to 4 at any given time. The reasons for this choice are as follows:

i) The mechanism is already available to adaptively send one-dimensional lines (i.e., the TAG bit).

ii) The top scan of a line of characters and the first white scan at bottom of a line of characters can often be encoded more efficiently by a one-dimensional algorithm.

iii) The choice of the algorithm for selecting which lines will be encoded one-dimension need not be standardized.

It was pointed out in the IBM paper that a code set can be adapted, real time, to the statistics of the run being encoded. The main criteria for the code selectivity is whether the run being encoded is following a horizontal mode run or a vertical mode run.

In an effort by 3M to simplify the code algorithm, it was assumed, but not statistically proven, that since K would adaptively select which lines would be encoded one-dimensionally, the majority of the correlated horizontal mode runs would be included within these lines. There would, therefore, be little reason to incorporate the multiple code set capability into the encoding algorithm.

2.3 Run Coding Method

The end of a run may be coded as the Horizontal Run Length, HRL, reference from the start of the run or by the relationship the end of the run has to a run on the previous line. This vertical mode method can be further sub-categorized as the current run ending directly Under the previous run, VMU, to the Right of the previous run, VMR, or to the Left of the previous run, VML.

Vertical Mode

The references show that the most likely occurrence is for transition elements at the end of vertically related runs to be directly aligned. Thus, this condition should be coded with a single bit. The references also state that the second most likely occurrences are that transition elements are displaced by one pel, left or right, from vertically related pels.

These two events have been given equal weighting and each assigned a three bit code. However, in typed script document, a displacement to the left (right) is likely to be followed by a similar displacement to the left (right). Therefore, the second most likely event is that two successive pel displacements are horizontally correlated (Reference Fig. 1). This event is given a two bit code and successive pel displacement, which are not horizontally correlated and are given a three bit code word.

Reference 4 shows that after considering vertical pel displacements of plus or minus one, little can be gained by extending the code to cover larger horizontal displacements. Thus, the choice was made to not code the vertical mode past one pel displacement. This is, of course, a value judgement which has been made based on simplification of implementation.

Horizontal Mode

This mode was the next most probable event and was given a 4 bit code followed by two modified Huffman run length codes.

The use of two successive run length codes was chosen since it has been shown that one horizontal run length tends to follow another horizontal run length.

2.4 Special Code Words

These code types are used in the coding scheme to specify conditions other than run coding. The only specific example studied here is the Pass Mode, PM, code. However, in general, any unique document feature would be handled with this general type of special coding parameter.

The PASS code is a code which could be optionally utilized by the encoder since the horizontal run length mode handles the same information (see Fig. 2). Furthermore, the run length mode would sometimes be more efficient than a series of pass words. The adaptive use of the pass code is thus a hardware implementation choice available to the designer of the encoder with no loss of interoperability.

2.5 Fill

This parameter takes the form of non-usable data inserted between encoded lines for the purpose of insuring the minimum scan line time. The use of a variable length string of zeros was retained.

2.6 Return to Control

This parameter indicates the end of the data encoding procedures and the transition to the control procedures. In Group 3 equipment, it is synonymous with the end of the document. The use of six successive EOL was retained since the mechanism is already in place to receive it.

3.0 CODE ALGORITHM

The code table is shown in Figure 3, some typical coding examples are shown in Figure 4, the coding flow diagrams are shown in Figures 5 and 6, and the code algorithm is described below.

The compression algorithm is comprised of two major steps. The first step is to decide whether to encode the current line one dimensionally or two dimensionally. The second step is to actually encode the current line.

The algorithm for selecting which lines to encode one dimensionally will vary greatly depending on the implementation. Thus, it was felt that the optimum, but more complex, algorithm should be tested. This consists of encoding each line using both one and two dimensional coding then transmitting the code with the fewest bits. This algorithm is described in the flow diagram in Figure 6 with the restriction that a one-dimensional line must be sent at least every four lines.

The two-dimensional algorithm flow diagram shown in Figure 6 does not utilize the PASS mode code option. This was done to simplify the encoding process. The algorithm then is reduced to the following logical flow.

Scan the line until a transition element, a_1, is

found. If it can be encoded by vertical correlation, then do so; if not, encode the accumulated run length and the succeeding run length using the standard (modified Huffman) one-dimensional scheme. Then continue to scan the line.

The following definitions are necessary in understanding the encoding flow diagrams.

a_0: Starting picture element on the current coding line which becomes the reference picture element for the beginning of a run. An a_0 element is always placed prior to the first picture element of each run.

a_1: Changing picture element on the current coding line, next to a_0. Element a_1 signifies the end of the a_0a_1 run. An a_1 element is always placed after the last picture element of each run.

a_2: Changing picture element on the current coding line, next to a_1.

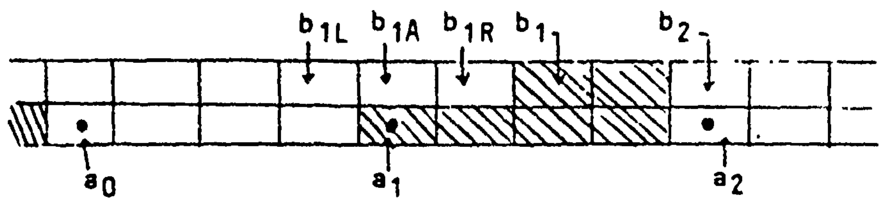
b_1: First changing picture element whose color is opposite to a_0, and which occurs on the reference line after the picture element just upon a_0.

b_{1A}: The picture element on the reference line which is directly above a_1.

b_{1L}: The picture element on the reference line which is one pel to the left of b_{1a}.

b_{1R}: The picture element on the reference line which is one pel to the right of b_{1a}.

D_2: Changing picture element coming after b_1 on the reference line.



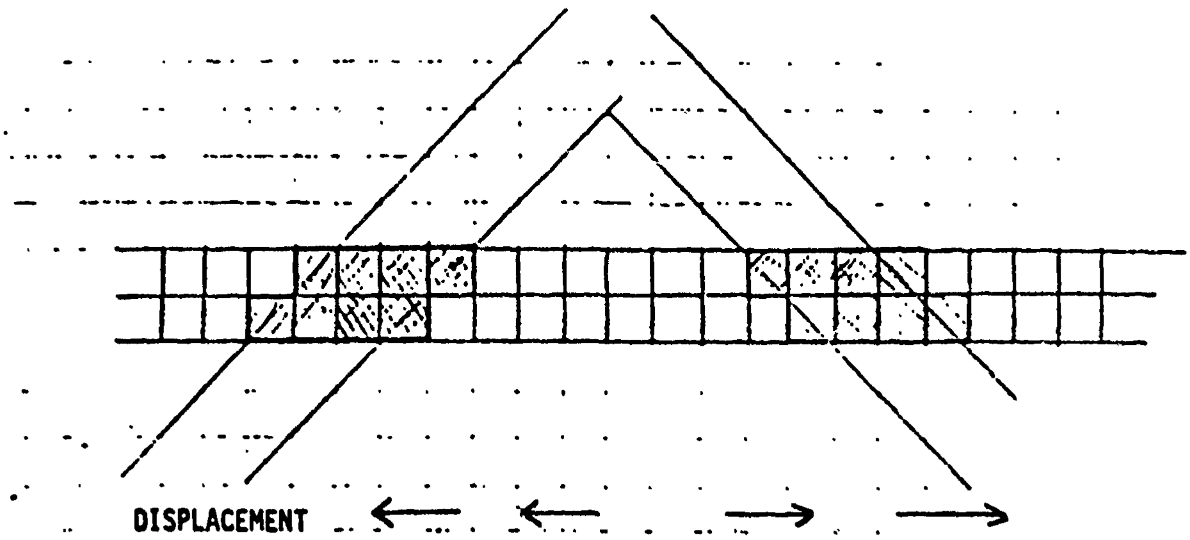
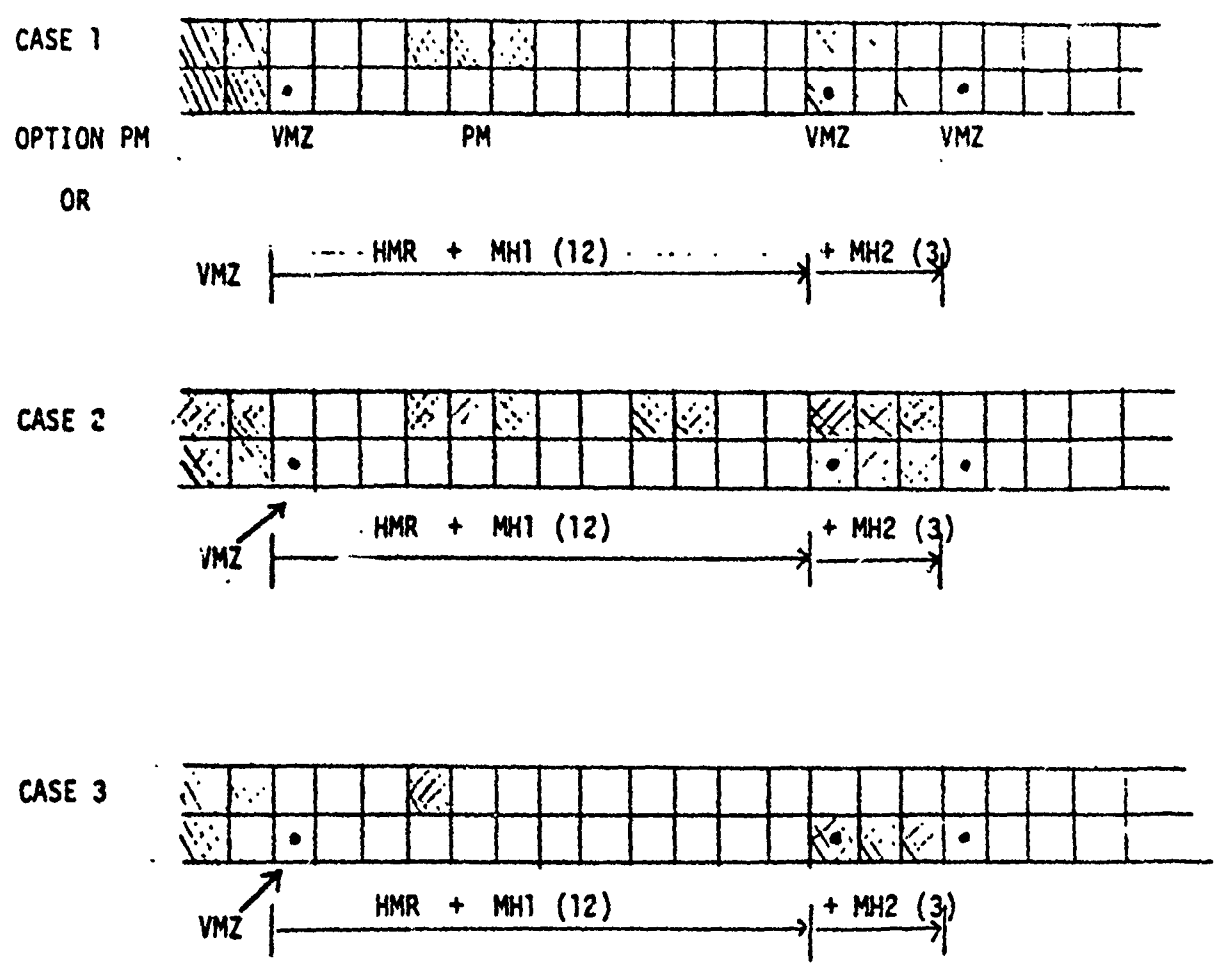


Figure 1 - Horizontally related vertical displacements



CASE 1 : OPTIONAL PM MAY BE SENT.

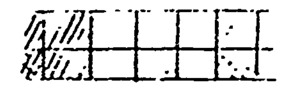
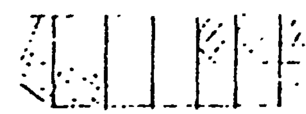
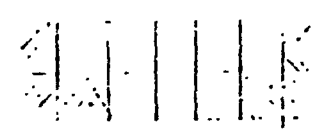
CASE 2 : THE FOUR TRANSITION ELEMENT ON THE REFERENCE LINE ELIMINATES THE PM OPTION.

CASE 3 : THE NECESSITY OF HORIZONTAL MODE RUN LENGTH CODING ELIMINATES THE PM OPTION.

Figure 2 - Explanation of the Optional PM Code

FIGURE 3

CODE TABLE

MODE TYPE	NOTATION	DEFINITION	EXAMPLE	CODE
Vertical Mode: zero displacement	VMZ	The transition element on the current scan line is directly under the transition element on the reference scan line.	 VMZ VMZ	0
Vertical Mode : Horizontally Correlated displacement	VMC	The transition element on the current scan line is displaced one pel from the transition element on the reference scan line. The direction of displacement, right or left, is identical to the preceding transition element on the current line.	 VMC or VMU VMC	01
Vertical Mode : Horizontally Uncorrelated displacement	VMU	The transition element on the current scan line is displaced one pel from the transition element on the reference scan line. The direction of displacement, right or left, is opposite from the preceding transition element on the current scan line.	 VMC or VMU VMU	001

CODE TABLE (CONT'D)

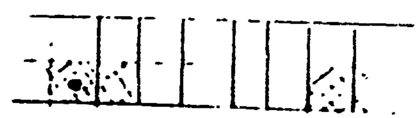
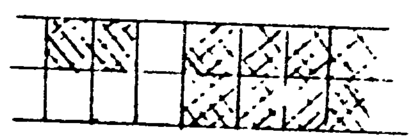
MODE TYPE	NOTATION	DEFINITION	EXAMPLE	CODE
Horizontal Mode : Run Length	HMR	The transition element of the current scan line is not correlated with a nearby transition element on the reference scan line. Thus, it is encoded relative to the previous transition element on the current scan line.		0001 +MH1, +MH2
Pass Mode	PM	The transition element on the current scan line is correlated to a nearby transition element on the reference scan line and will be so encoded. However, there is an extraneious pair of transition elements on the reference scan line which must be ignored.		00001

Figure 3

(2525)

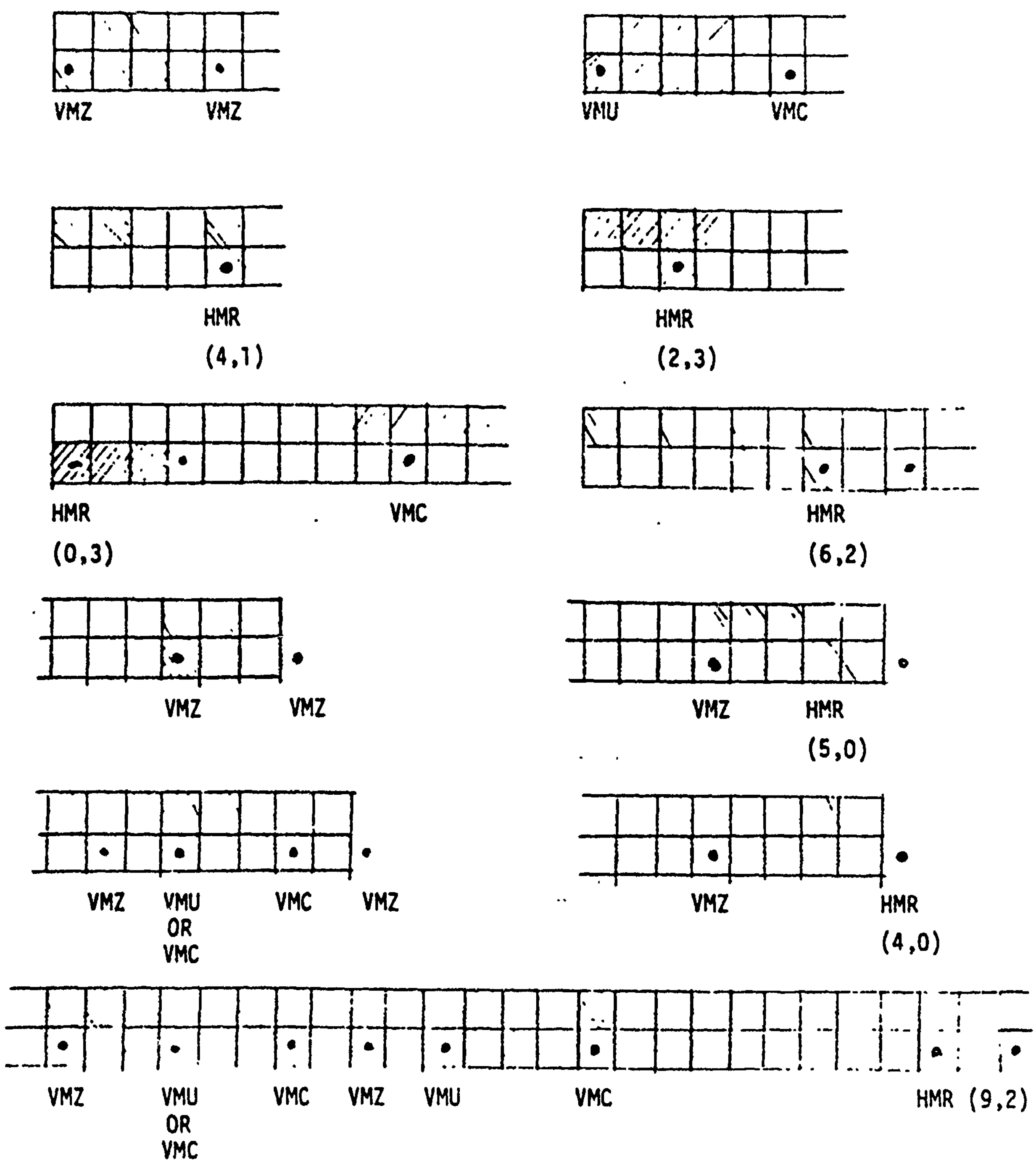


Figure 4 - Typical coding examples

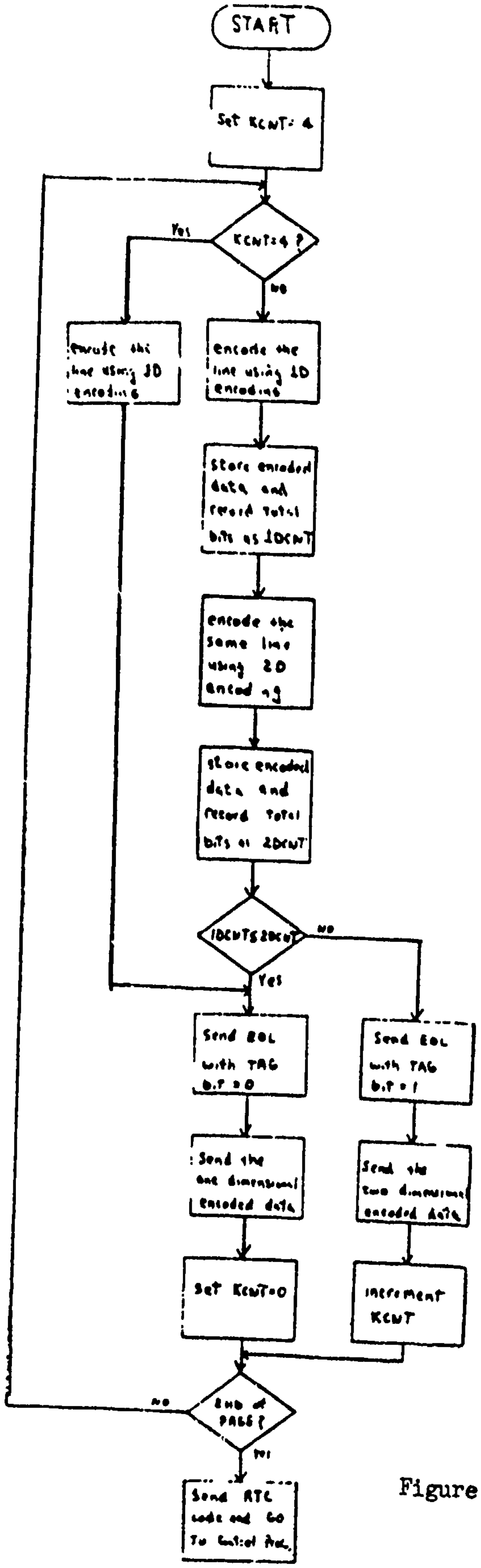


Figure 5 - Flow diagram for compressing on facsimile document

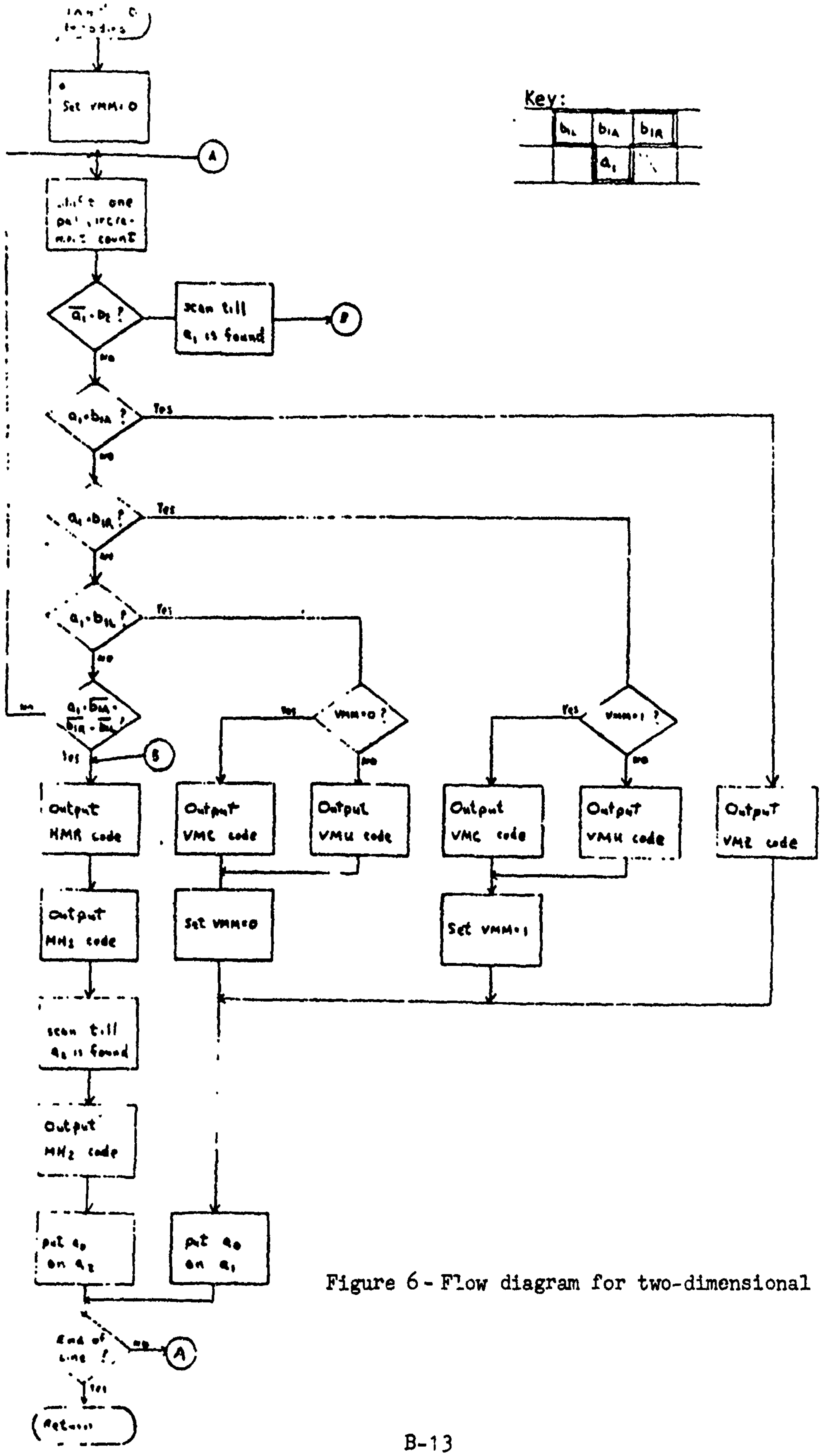


Figure 6 - Flow diagram for two-dimensional coding

REFERENCES

1. Japan — COM-XIV No 42-Period 1977-1980
   Proposal fc· Draft Recommendation of
   Two-Dimensional Coding Scheme

2. IBM — COM-XIV Delay Doc. ≠9-Period 1977-1980
   Proposal for Two-Dimensional Coding
   Scheme

3. Nippon Telegraph and Telephone Public Corp. — Temporary Document No. 8, Meeting of
   the Special Rapporteur Group on Group 3
   Machine, Paris, November 1976
   Proposal on Coding Scheme Standardization
   for Group 3 Machines

4. Dr. Joan L. Mitchell, Gerald Goertzel — IBM Research Report RC 7499 (≠32376)
   Two-Dimensional Facsimile Coding Scheme

International Telegraph and Telephone
Consultative Committee
(CCITT)

Period 1977-1980

Corrigendum to
COM XIV-No. 74-E

Original : English

Question 2/XIV

Date : July 1979

CORRIGENDUM TO STUDY GROUP XIV - CONTRIBUTION No. 74

SOURCE : 3M COMPANY

TITLE : CORRECTION TO THE PROPOSED TWO-DIMENSIONAL CODING ALGORITHM

1. Introduction

In an effort to complete Contribution COM XIV-No. 74 by the March deadline, some errors in the original text were not found during proofreading. The corrections for these errors are listed below.

Section 3.0, 3rd Paragraph, 8th line: Change Figure 6 to Figure 5.

Figure 3: All of the codes are inverse polarity. The coding should be:

MODE	CODE
VMZ	1
VMC	1 0
VMU	1 1 0
HMR	1 1 1 0 + MH1, MH2
PM	1 1 1 1 0

Figure 6: The mnemonic VMM means: Vertical Mode Memory.

APPENDIX C

CCITT STUDY GROUP XIV

Contribution No. 64

Source: IBM Europe

International Telegraph and Telephone
Consultative Committee
(CCITT)

Period 1977-1980

COM XIV-No. 64-E*)

Original : English

Question : 2/XIV

Date : 19 January 1979

STUDY GROUP XIV - CONTRIBUTION No. 64

SOURCE: IBM EUROPE

TITLE: PROPOSAL FOR TWO-DIMENSIONAL CODING SCHEME.

1.0 Introduction

Draft Recommendation T.4, generated at Study Group XIV meeting Nov. 14-18, 1977, provides for the possible extension of the (standard) one-dimensional coding scheme to a two-dimensional scheme. This was to be the subject of further study. The contribution contained in the following pages describes such a two-dimensional coding or compression scheme, based on the one-dimensional scheme of Draft Recommendation T.4.

The proposed compression scheme contains the one-dimensional scheme as a subset and requires that the first line of a page be encoded in this manner. When required, the entire page may be sent as the one-dimensional scheme. A given machine can readily operate with either the one- or two-dimensional scheme as dictated by the capability of the receiver.

Taking advantage of the vertical correlations between scan lines as well as the horizontal redundancy results in significant improvements in compression. With two-dimensional coding the compressed data does not double when the vertical definition is doubled. This improves the speed at which the optional higher vertical definition can be transmitted. For some applications the higher quality with the greater vertical definition may be essential.

Ease of hardware implementation, compression efficiency, speed of processing, error sensitivity, and future extendability to higher resolutions are all important in selecting the two-dimensional option.

*) Contribution retardée publiée seulement dans la langue reçue (anglais), sans traductions supplementaires, conformément aux dispositions de la Résolution N° 1, paragraphe III.4.d) adoptée par la VIe Assemblée (1976).

Late Contribution published only in the language received (English), without further translations, in accordance with Resolution No.1, paragraph III.4.d)VIth Plenary Assembly (1976).

Contribución retardada publicado sólo en el idioma en que se recibió (inglés), sin otras traducciones de conformidad con la Resolución N.°1, párrafo III.4.d), VI Asamblea Plenaria (1976).

(2164)

Higher resolution and speed requirements should not make the data compression algorithm obsolete.

The following two-dimensional data compression scheme is proposed as the two-dimensional option for Group 3 machines.

2.0 Two-Dimensional Coding

The two-dimensional scheme is a line-by-line coding method which requires the storage of one scan line for use as the history line for coding the next line. It is a natural extension of the one-dimensional data compression standard. Those runs which are highly vertically correlated are referenced to the history line. The rest are coded with the Modified Huffman Codes of the one-dimensional scheme. Improvements over traditional line-by-line coding schemes are obtained by making efficient use of the code words and by allowing dynamic interpretation of some code words. The end of line (EOL) code contains the same unique string of ten zeros to use for resynchronization purposes as the one-dimensional standard. Fill zeros can be used to obtain the minimum (permissible) transmission time. This scheme can code any length line. Thus, it allows for future extensions of coding to higher resolutions or wider pages.

More detailed descriptions of the coding method are given in Appendix 1 along with some coding examples. Appendix 2 illustrates how the scanned image can be processed synchronously by examining the current scan picture element (pel) and two pels above it on the history line. State diagrams are shown for processing the scanned data in one step and then generating the code words in a second step which can be in process at the same time.

3.0 Compression Results

Appendix 3 gives tables of the compression achieved with the two-dimensional coding for three sets of the eight CCITT test documents at the normal vertical definition and for two sets at the higher vertical definition. The effect of two treatments of the history line at the right edge is also shown.

Two-dimensional coding schemes based on line-by-line coding risk having errors made in one line propagate into succeeding lines. This error propagating effect can be restricted by interleaving lines encoded by the two-dimensional scheme with lines encoded by the one-dimensional scheme. The interleaving can be done systematically, a one-dimensional line every Kth line; however, as proposed here, the bit stream carries information as to how each line is encoded so that the transmitter may either inject one-dimensional encoding in any desired manner or may be required to do so periodically.

For the normal vertical resolution the three sets of documents have an average transmission time ratio of 87% when one- and two-dimensional coding is alternated for successive scan lines as compared to the standard one-dimensional coding. At the higher definition the average transmission time ratios for two sets is 70% when only every fourth line

is coded one-dimensionally. The average ratio improves to 65% for the normal definition and 52% for the higher definition if one-dimensional coding is not required periodically.

4.0 Conclusion

The proposed two-dimensional extension of the one-dimensional coding standard gives significant improvements in compression over the one-dimensional scheme. Flexibility in the choice of the frequency of one-dimensional coding and in the use of EOL's permits trade-offs between compression efficiency and protection against error propagation to be made according to the local error environment.

APPENDIX 1

TWO-DIMENSIONAL CODING SCHEME

The following two-dimensional data compression scheme which is an extension of the one-dimensional standard is proposed as the Group 3 machine two-dimensional option. The same data compression scheme with a slightly different treatment of a document's right edge can be used for error free environments.

1.0 Error Propagation

As mentioned previously, for reasons of restricting error propagation, one-dimensional coding can be used for the first of every K lines. The parameter K can be chosen to fit the local environment. For Group 3 machines it is recommended that K = 2 for the normal vertical definition and K = 4 for the higher resolution.

2.0 One-Dimensional Coding

The first line must be sent one-dimensionally. The Modified Huffman Codes (MHC) of the present one-dimensional standard are used. They depend upon the run length count (CT) and whether the run is black (B) or white (W).

3.0 Two-Dimensional Coding

The two-dimensional scheme is a line-by-line coding method which requires the storage of one scan line for use as the history line for coding the next line. It is a natural extension of the one-dimensional data compression standard. Those runs which are highly vertically correlated are referenced to the history line. The rest are coded with the Modified Huffman Codes of the one-dimensional scheme. Improvements over traditional line-by-line coding schemes are obtained by making efficient use of the code words and by allowing dynamic interpretation of some code words. The details of the scheme are specified below.

3.1 Transition Elements

A transition element is usually defined as the pel following a change of color. For purposes of illustration the following labels will be used:

S - Transition element which starts the run.

C - Transition element which ends the current run.

H - First transition element on the history line (if it exists) which is the same color as C and to the right of S.

Figure 1 shows S, C, and H for two runs.

If an end of line code is to follow the data for the current scan line, a hypothetical C is assumed after the rightmost pel to force termination of the run.

For Group 3 machines (in which an EOL code follows the coding of every scan line) the history line is extended by two hypothetical pels. If the final history pel is white these pels are a black-white pair. For a final black pel they will be a white-black pair. This guarantees that there will always be an H for the final run because both types of transition occur on the right hand edge.

3.2 Vertical Reference Coding

If C is directly under H, the run is coded as an error of zero (EZ) in vertical alignment. If C is one position to the left of H, the run is identified as an error of minus one (EM) when compared to H. Similarly if C is one pel to the right of H, then it is called an error of plus one (EP). Some examples are shown in Figure 2A.

If C is not within one pel of H, then the run is identified as a run length feature. See Figure 2B for an example. In order to distinguish the run length codes from the vertical reference codes, a run length prefix (RL) precedes the count encoded with MHC (Modified Huffman Codes).

The runs which must be sent as run length features are not independently distributed among the vertically aligned runs. About half of them immediately follow another run length feature. Even though EZ is most likely in general, after a run length feature another run length feature is most probable. This strong correlation can be used to achieve better compression by making the code assignments for EZ and RL conditional on the previous feature.

The Code Table is listed below for the PL prefix and the vertical reference codes.

Code Table

	Following run length	Following EOL, FZ, FP or FM
RL-prefix	1	01
EZ	01	1
EM	001	001
EP	000	000
		0001 *

* Even numbered EP's in a series of EP's have an extra 1. This extra 1 is required to prevent a false EOL code in the event that several EPs in a row are encountered.

3.3 Efficient Use of Code Words

An additional improvement in compression is achieved by efficient use of the code words. The run length to be coded is shortened by not counting those pels in the run which are directly under H or one or two pels to the left. Thus the counter (CT) will sometimes have two or three counts less than the actual run length and the MHC for CT may be shorter. Small circles in Figure 3 show those pels which are not counted. If any of those pels had been the final pel in the run, then vertical reference codes would have been sent rather than run length codes. The decoder in turn recognizes the same pels and does not decrement the counter for the positions directly under H or one or two to the left of H. This usually gives a systematic 2-3% improvement in compression.

3.4 Extention To Lines Longer Than 2560 Pels

Every time CT (run length count) = 2560, a carry code (same bit pattern as the makeup code word for 2560) is generated. S is then moved past the last pel counted and a new run of the same color is started. If this new S is identical with C, then a run length of zero for the old color is sent. Carry is the only makeup code which is not necessarily followed immediately by a terminating code. During two-dimensional coding a carry can be followed by a vertical reference code, another run length code including a carry, or an EOL.

4.0 End Of Line Codes (EOL)

EOL codes contain the same unique string of at least ten zeros followed by a one that is in the one-dimensional EOL. However, the optional longer string of at least eleven zeroes followed by a 1 is always used. A tag bit is added to indicate which type of coding will follow. A tag of 0 indicates one-dimensional and a tag of 1 specifies two-dimensional coding.

During two-dimensional coding a run length prefix must precede the EOL to distinguish it from EP's. A string of EP's is prevented from generating eleven or more zeros in a row by appending a 1 to the even numbered EP's.

End of line codes are required to precede the first line (which must be coded one-dimensionally) and must also occur whenever a switch is being made between one- and two-dimensional coding. In addition, for Group 3 machines it is expected that EOL's will be required on every line.

After an EOL the first code word represents a white run. If the first pel is black, a white run length of zero must be sent to specify the color change. When two-dimensionally coding, an EZ is usually more likely than a run length code: so it is given the shorter code word after an EOL.

5.0 Fill

Fill zeros can be used to obtain the minimum (permissible) transmission time per line. The extra zeros are inserted into the string of zeros in the EOL codes. An EOL must be used if fill is needed.

6.0 Return To Control (RTC)

The return to control sequence is maintained as six end of line codes. Each EOL should indicate one-dimensional coding to follow.

7.0 Error Free Environments

In error free environments or where retransmission is possible, EOL's may add unnecessary extra bits. The coding scheme allows for considering the scanned image as one continuous bit stream in which the rightmost pel on a line is followed immediately by the leftmost pel on the next scan line. Figure 4 illustrates the sequence of pels. While this wraparound capability may not be desired in Group 3 apparatus, the capability may be very useful in the proposed Group 4 apparatus.

The MHC's require knowledge of the color of the run for correct decoding. For the one-dimensional coding scheme every line starts with a white code word. (A run length of zero for the white run is sent if the line actually starts with a black pel.) EOL's on every line make it possible for the decoder to identify the starts of new lines without having to first complete reconstruction of the last scan line. Without the EOL's it is possible for two white code words to occur together in the compressed data stream. Parallel (and independent) processing of the compressed data and reconstruction of the image is severely restricted if the color of the next code is dependent upon whether the preceding code word represented the final run on a line. So in environments where the EOL's are not used, the wraparound technique guarantees that the color of the code words alternates (with the exception of the carry feature which is known to be followed by the same color). This allows coding of the compressed data into features to be independent of the position of the runs on the scan lines.

Figure 5 shows determination of the features using both techniques. The hypothetical history pels are dotted in Figure 5A. The small circles show which pels are not counted. The features are explained in Appendix 2. In both cases the first line is coded one-dimensionally and ended with an EOL. In Figure 5A EOL's are required for each line while in Figure 5B EOL's are only used to switch coding schemes.

APPENDIX 2

STATE DIAGRAMS

The process of determining the code words to represent the scanned data can be illustrated by state diagrams in two steps. The first step synchronously uses three bits (one pel on the current line and two above it on the history line) to determine the next state. Every time a transition element is found, a feature is output to the second step. The more common features are EZ, EP, EM, RLW, RLB, DRLW, and DRLB. (A 'D' prefix on features and states indicates one-dimensional coding.) These features are used as the input to the second step. Since the second step remembers the last code word type, the correct bit patterns can be generated for any sequence of features.

1.0 Inputs to Encoder Step 1

Three bits of data are needed to determine the next state. They are the current pel (CP), the history pel (HP) which is immediately above CP on the history line, and the history pel look ahead (HPLA) which is one pel to the right of HP. Figure 6 shows the positions of CP, HP, and HPLA as CP moves from the one scan line to the next. In Figure 6A CP is the third position from the end of the second line. Above it is HP and above to the right HPLA. In Fig. 6B the next pel is being processed. For Fig. 6C CP is now the final pel in the line. The HP is still above it. Since hypothetical history transitions (t) are to be introduced for Group 3 machines, HPLAt is a hypothetical pel to the right of HP. The hypothetical pels are shown as dotted boxes. If instead the history is to be treated as a continuous (c) stream, HPLAc has moved to the next scan line (which is the same scan line that the CP is on). Figure 6D shows HPt, HPLAt, HPc, and HPLAc if the run is forced to terminate for an EOL. The HPc and HPLAc are the first two pels on the scan line being completed. The CP for generating a hypothetical transition to terminate the run is also shown. Figure 6E shows the condition when the new scan line, namely the third scan line, is started and Figure 6F shows processing the next pel.

2.0 State Diagram for Encoder Step 1

The state diagram for the Encoder Step 1 is illustrated in Figure 7. The three input bits are CP, HP, and HPLA respectively. A white pel is specified by a '0' and a black pel by a '1'. Dashes in the state diagram indicate 'don't care' bits which can be either a 0 or 1. Triangles indicate when a different path will be taken according to whether CT = 2560 or CT < 2560. The single asterisk (*) indicates when to increment the counter. A double asterisk (**) designates that the counter is reset to 1. States are shown in circles while the outputs (which may have the same name) are in boxes.

As long as CP is white, the next state will be one of the white states on the left side of the state diagrams for Step 1. For a black CP, the next states are on the right side of the diagrams. Each time the color changes or CT = 2560, some output is generated for use in Step 2.

2.1 One-Dimensional Coding

Processing of the scanned pel starts with one-dimensional coding in the START 1-D state in Figure 7A. A white pel causes a transition to the one-dimensional run length white (DRLW) state. As long as CP remains white, the counter will be incremented and the state stays the same. A change of CP to black outputs a one-dimensional run length white feature (DRLW) and causes a transition to the one-dimensional run length black state (DRLB). This process continues until the end of the line. If an end of line code is desired, the run is forced to terminate and the final run length feature is followed by an end of line feature which specifies continuing one-dimensional coding (DEOL1) or switching to two-dimensional coding (DEOL2). (The end of line sequence is not shown in the state diagrams.)

The diagram also shows what happens when the run length counter reaches 2560 without a transition. In that case a one-dimensional carry feature (DCRY) is output and a new run of the same color is started.

After a DEOL2 (which specifies two-dimensional encoding) the START 2-D state is entered (See Figure 7B).

2.2 Two Dimensional Coding

If CP is black in the START 2-D state, then a run length zero white (RLOW) feature is output in order to indicate the color change. For white runs, the first time HP is white and HPLA is black locates H. For black runs, the significant combination is a black HP and a white HPLA. Once H has been found it is no longer necessary to consider the history line until C occurs.

If H does not occur immediately after S, then the run will begin in the start white (SW) state or start black (SB) state and will continue in those states until either H is found or the run length terminates. It is possible, if no EOL was used on the last line, to get to a count of 2560 before C. In that case a carry is sent out (CRY) and a new run of the same color started. If C occurs just before H, then an EM codes that particular run. Otherwise a run length white (RLW) or run length black (RLB) is sent. When H is reached on the history line, then the next state is the white EZW or black EZB state. If the run continues, it goes to the EP (EPW or EPB) state and finally into the run length (RLW or RLB) state. If a termination occurs in the EZW or EZB states, then the feature will be an EZ. If it happens one pel later while in the EPW or EPB states, the output will be EP.

From the run length states (RLW and RLB) the output is usually a run length feature. However if the counter reaches 2560, a carry (CRY) feature is generated. After outputting CRY a new run is started. If the run should be exactly 2560, then a carry must be followed by a run length of zero (CROW and CROB).

It is necessary in a two-dimensional coding scheme to have history data that would be the same as that seen in the decoder so that interpretation of the code is independent of the edge of the image. Therefore when run lengths are forced to terminate at the end of the scan line so that an EOL can be inserted, for Group 3 machines the HP and HPLA are hypothetical pels. When EOL's are not required on every line, the HP and HPLA used are the same as for the first pel on the next line.

The EOL for use at the end of two-dimensionally coded lines is either an EOL1 which specifies one-dimensional coding next or an EOL2 which specifies continuing with two-dimensional coding. The run length prefix precedes the string of at least eleven zeros followed by a 1. The tag bit is a 0 for EOL1 and a 1 for EOL2.

2.3 Return to Control

Six EOL's are generated for the return to control sequence. If the final line were two-dimensionally encoded, the features would be EOL1, DEOL1, DEOL1, DEOL1, DEOL1, and DEOL1. For one-dimensional coding six DEOL1's are required.

3.0 Examples of Features

Figure 5 gives some examples of features which would have been determined by Step 1. Circles indicate pels which are not counted. The dotted pels in Figure 5A are the hypothetical history pels. Figure 5B shows how the wraparound decreases the number of features needed to characterize the image.

4.0 State Diagram for Encoder Step 2

The state diagram for the second step of the coding process which translates the features into code words is shown in Figures 8A, 8B, and 8C. As defined below the states PR, PE, and PEP indicate the previous type of code word. If the previous code was a run length code (which includes the carry), Step 2 will be in the PR state. If it was an EZ, EM, or end of line code, it will always be in the PE state. Following an EP there are two possible states, either PE or PEP depending upon whether the EP was the even or odd numbered feature in the sequence of EPs. The input to Step 2 is a feature which has been output from Step 1.

(2164)

APPENDIX 3

COMPRESSION RESULTS

Compression results are reported in Tables 1 to 6 for three different scannings of the eight CCITT test documents. The A set was used for the Graphics Coding Contest of the 1976 Picture Coding Symposium (Asilomar, California, January 1976). The documents were digitized for 1728 pels per line by 2128 lines, nominally 8 pels/mm x 8 lines/mm (8 x 8). The sets B and C were obtained from the French PTT. The B set was digitized at 1680 pels per line by 2376 lines (8 x 8). The C set was scanned with 1680 pels per line by 1180 lines, nominally 8 x 4. The two higher vertical definition sets were used to simulate 8 x 4 definition by compressing only the even numbered lines. Table 1 summarizes the specifications for the images.

Table 2 lists the results as average transmission times per document set for one-dimensional coding at 4800 bits/sec. The EOL used for the CCITT standard was eleven bits (00000000001). For K = 1 the EOL on each line was thirteen bits (0000000000010). The 5 msec and 10 msec results include sufficient fill bits to obtain those minimum transmission times per line (24 bits and 48 bits respectively). The average transmission times can be converted into average compression factors by dividing the total pels by 4800 x time.

The label WRAP indicates that the images were treated as a continuous bit stream with wraparound. Runs are terminated only if an EOL had to be inserted into the compressed data stream in order to change coding from the one-dimensional to the two-dimensional or from the two-dimensional to the one-dimensional schemes. The EOL's before the first line (which is coded one-dimensionally) and the RTC sequence are included. For K = 1 there is no switching; so these are the only EOL's. For K = 2 there will be an EOL after every line. For K > 2 an EOL will precede and follow each one-dimensionally coded line. No fill bits are used in wraparound results.

The labels NO EOL, 5 msec, and 10 msec always indicate that the final run on each line terminated with a hypothetical C transition element. For NO EOL the fill bits and EOL have been removed.

Removing the EOL's and fill needed for 5 msec saves about 5% for one-dimensional coding. However, the wraparound technique saves another 5% because the margins on the right and left edges become one long run instead of two.

The average transmission times are listed in Table 3 for the case in which only the first line is coded one-dimensionally and the rest of the image is coded two dimensionally (K = infinity). The results obtained by terminating the history line with hypothetical pels on the right edge are labeled TRANS. (About 90% of the time this introduces a hypothetical H transition element.) The history line can also be treated as a continuous bit stream (CONT.) even though an EOL is required on every line. Having a continuous history rather than

(2164)

transitions on the right edge gives about 10% less compression for the NO EOL case, but less fill is needed, so there is only about a 5% difference for the 5 msec case. If EOL codes are not needed, then wraparound (WRAP) can give compression within 2% of NO EOL with hypothetical history transitions (TRANS.) (In addition to the EOL's and fill bits, the EOL run length prefix and the extra 1's in a sequence of EP's are not included in the two-dimensional NO EOL results.)

Tables 4 and 5 show the performance for Group 3 machines for K = 2 and K = 4. The 10msec CONT. columns for K = 2 and K = 4 were obtained by interpolation. The rest of the entries came from simulations of the compression or actual encodings of the images.

Table 6 collects together the average transmissions times for k = 2 and k = 4 (TRANS.) for the proposed scheme for Group 3 machines and the results using wraparound for k = infinity along with the current CCITT one-dimensional coding scheme. The ratios of each two-dimensional coding scheme to the one-dimensional scheme are also tabulated. The average transmission time ratio for the normal definition (K = 2,TRANS.) is 87%. For the higher definition this ratio is 70% (K =4,TRANS.).

Table 1. Document Description
Eight CCITT Test Documents in Each Set

DOC. SET	NOMINAL DEFINITION	LINES DOC.	PELS/ LINE	TOTAL PELS	SOURCE
A8	8 x 8	2128	1728	3677184	Graphics Coding Contest
B8	8 x 8	2376	1680	3991680	French PTT 8 x 8
A4	8 x 4	1064	1728	1838592	Half of A8 lines
B4	8 x 4	1188	1680	1995840	Half of B8 lines
C4	8 x 4	1188	1680	1995840	French PTT 8 x 4

Table 2. One-Dimensional Coding

Average Transmission Time (Seconds)
(4800 bits/sec)

DOC. SET	NO EOL	CCITT 1-D 5msec	CCITT 1-D 10msec	K = 1 5msec	K = 1 10msec	WRAP
A8	89.0	93.9	95.8	94.8	96.5	83.9
B8	107.1	112.6	115.2	113.5	116.0	101.9
A4	44.5	47.0	48.0	47.4	48.3	42.0
B4	53.5	56.3	57.6	56.8	58.0	51.0
C4	51.6	54.4	55.9	54.8	56.2	49.2

Table 3. Two-Dimensional Coding K = infinity

Average Transmission Time (Seconds)
(4800 bits/sec)

DOC. SET	NO EOL TRANS.	NO EOL CONT.	5 msec TRANS.	5 msec CONT.	10 msec TRANS.	10 msec CONT.	WRAP
A8	50.4	56.4	58.1	62.9	61.4	64.6	50.8
B8	56.8	62.6	65.6	70.0	69.5	72.6	56.5
A4	31.8	34.7	35.6	37.9	37.1	38.7	31.9
B4	35.4	38.2	39.6	41.7	41.5	43.1	35.0
C4	34.3	37.3	38.6	40.9	40.5	42.3	34.5

Table 4. Two-Dimensional Coding K = 2

Average Transmission Time (Seconds)
(4800 bits/sec)

DOC. SET	5 msec TRANS.	5 msec CONT.	10 msec TRANS.	10 msec CONT.	WRAP.
A8	76.4	78.8	78.9	80.6	78.8
B8	89.4	91.7	92.6	94.2	91.7
A4	41.5	42.7	42.7	43.5	42.7
B4	48.2	49.3	49.7	50.5	49.3
C4	46.7	47.8	48.3	49.2	47.8

Table 5. Two-Dimensional Coding K = 4

Average Transmission Time (Seconds)
(4800 bits/sec)

DOC. SET	5 msec TRANS.	5 msec CONT.	10 msec TRANS.	10 msec CONT.	WRAP.
A8	67.3	70.8	70.1	72.6	64.9
B8	77.5	80.8	81.0	83.4	74.2
A4	38.6	40.3	39.9	41.1	37.3
B4	43.9	45.5	45.6	46.8	42.2
C4	42.7	44.4	44.4	45.8	41.2

Table 6. Ratio of Two-Dimensional to One-Dimensional Transmission Times

DOC. SET	1-D sec	K = 2 sec	RATIO	K = 4 sec	RATIO	WRAP sec	RATIO
A8	93.9	76.4	0.81	67.3	0.72	50.8	0.54
B8	112.6	89.4	0.79	77.5	0.69	56.5	0.50
A4	47.0	41.5	0.88	38.6	0.82	31.9	0.68
B4	56.3	48.2	0.86	43.9	0.78	35.0	0.62
C4	54.4	46.7	0.86	42.7	0.79	34.5	0.63

(2164)

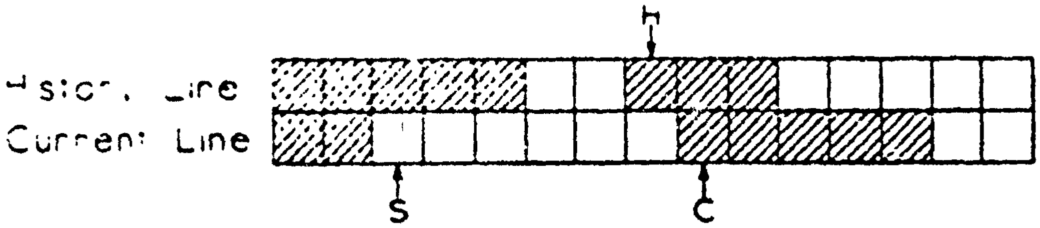


Figure 1A. Transition elements for the white run.

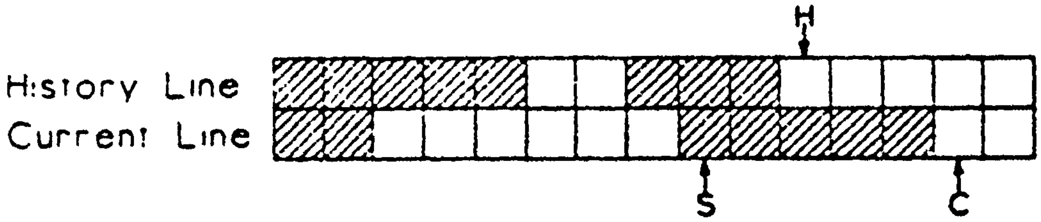


Figure 1B. Transition elements for the next (black) run.

Figure 1. Transition elements

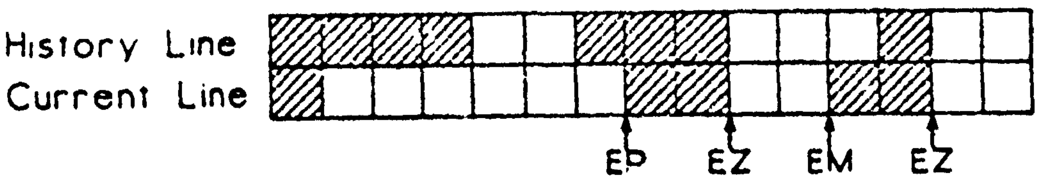


Figure 2A. Examples of Vertical Reference Features.

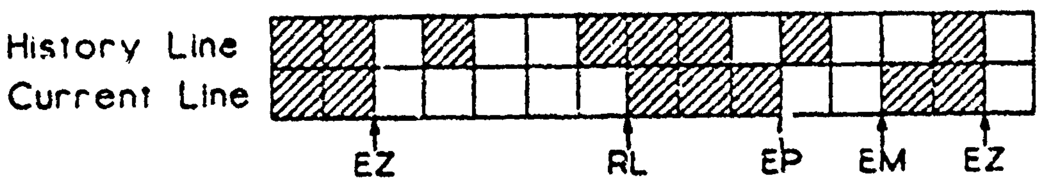


Figure 2B. Example with a Run Length Feature.

Figure 2. Features

(2164)

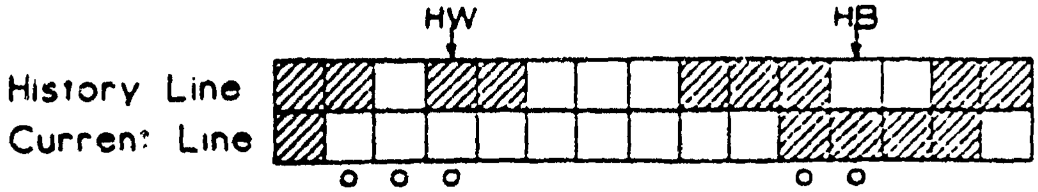


Figure 3. Circles show pels which are not counted.

Scan Line					
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20

Figure 4A. Four lines of five pels each.

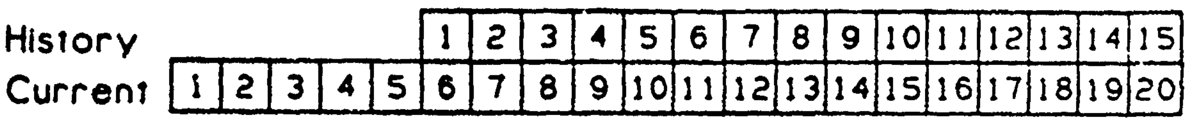


Figure 4B. Continuous pel stream.

Figure 4. Wraparound

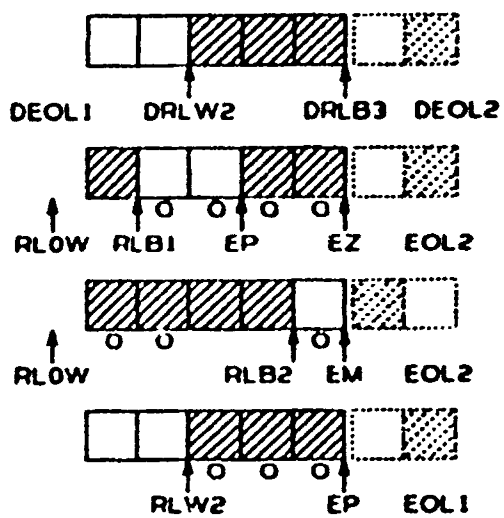


Figure 5A. EOL every line.

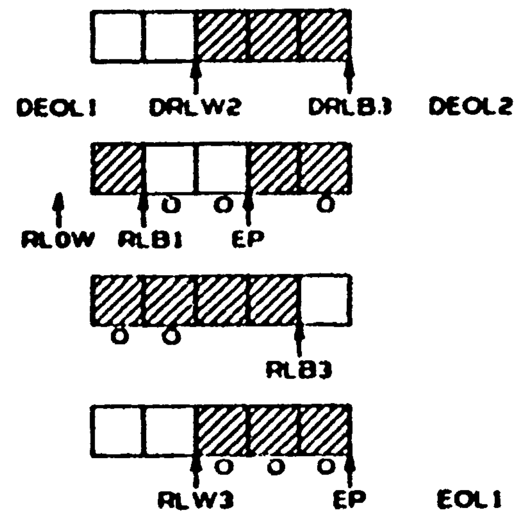


Figure 5B. Wraparound.

Figure 5. Examples of Feature Determination.

(2164)

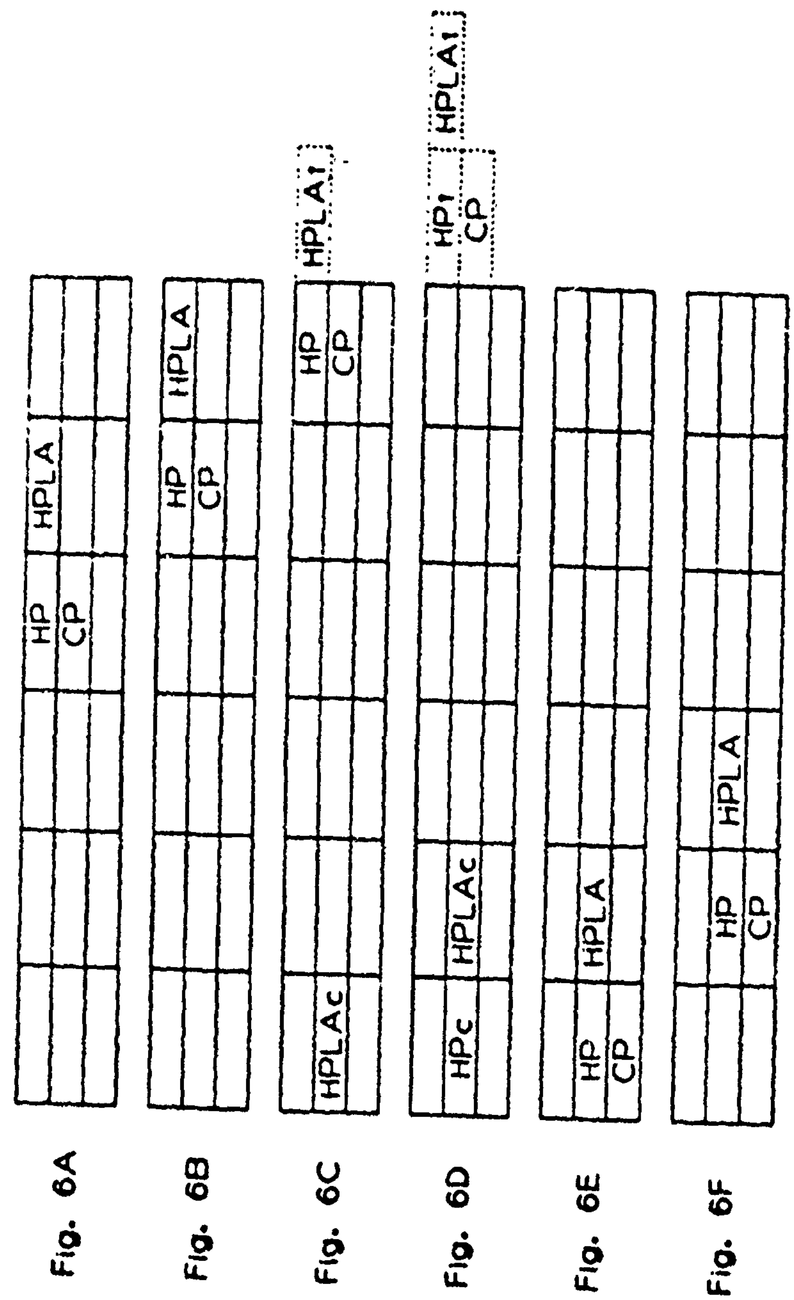


Figure 6. INPUTS TO ENCODER STEP 1

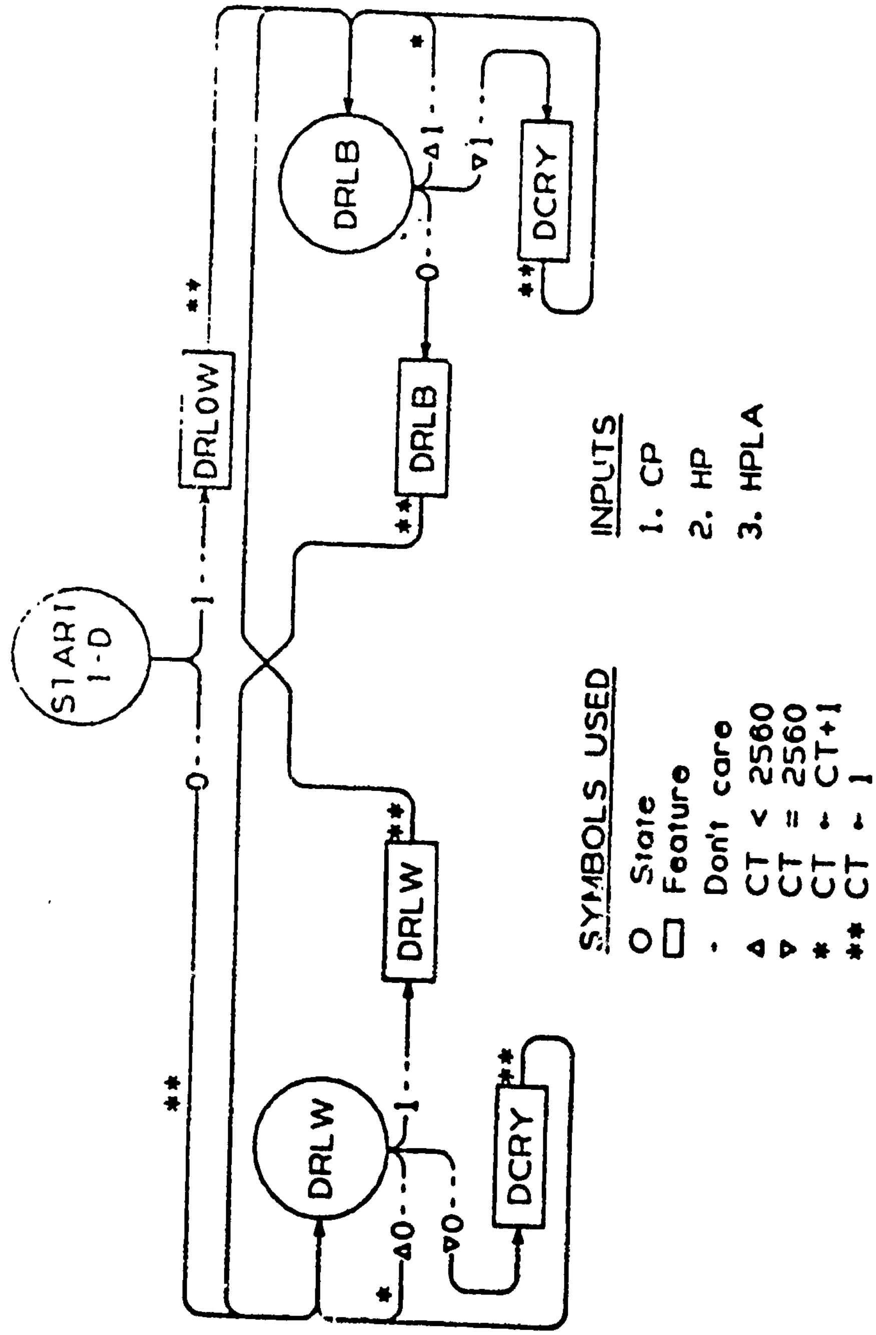


Figure 7A.
STATE DIAGRAM FOR ENCODER STEP 1
(ONE-DIMENSIONAL MODE)

(2164)

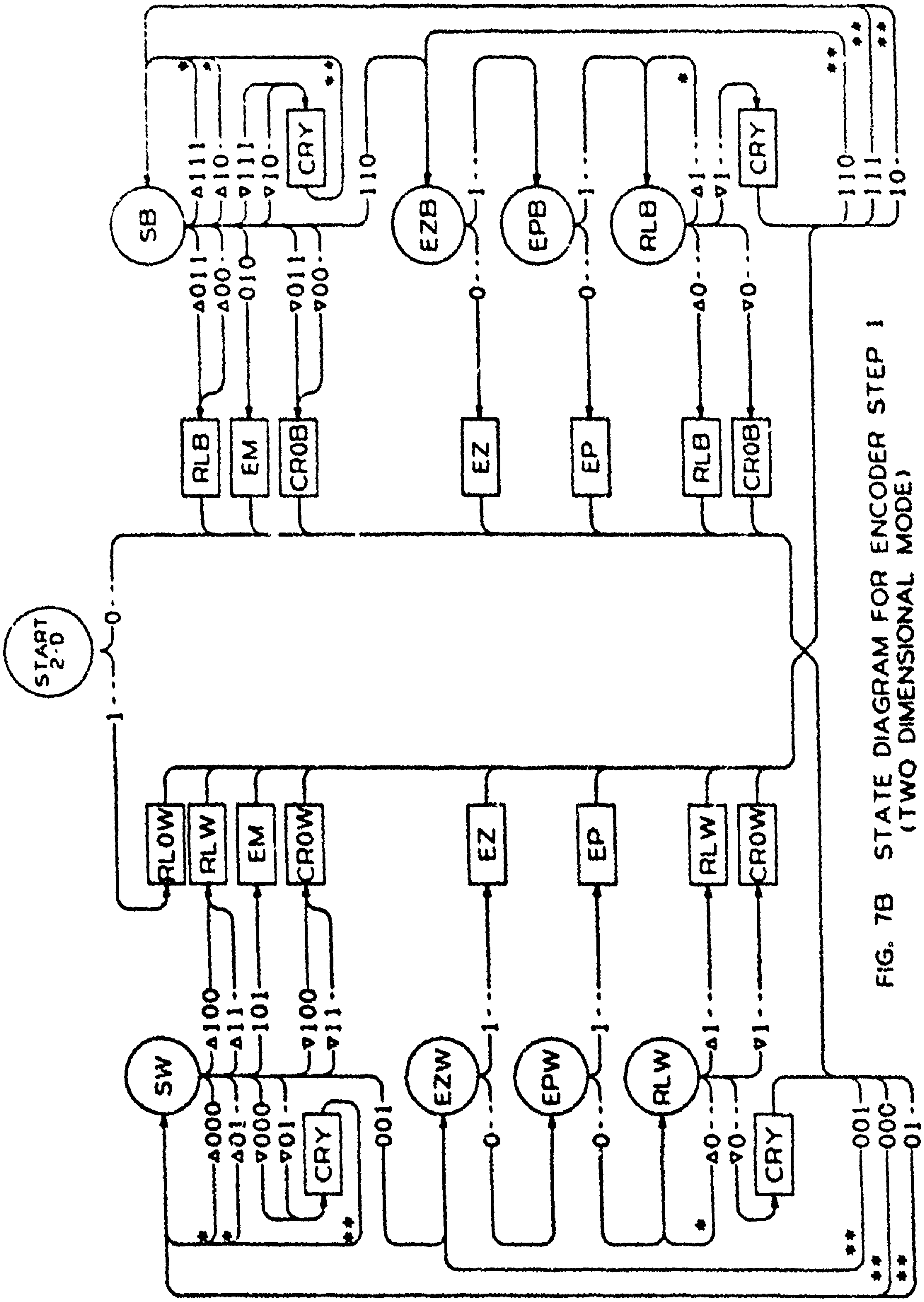


FIG. 7B STATE DIAGRAM FOR ENCODER STEP 1 (TWO DIMENSIONAL MODE)

Figure 8A
STATE DIAGRAM FOR
ENCODER STEP 2

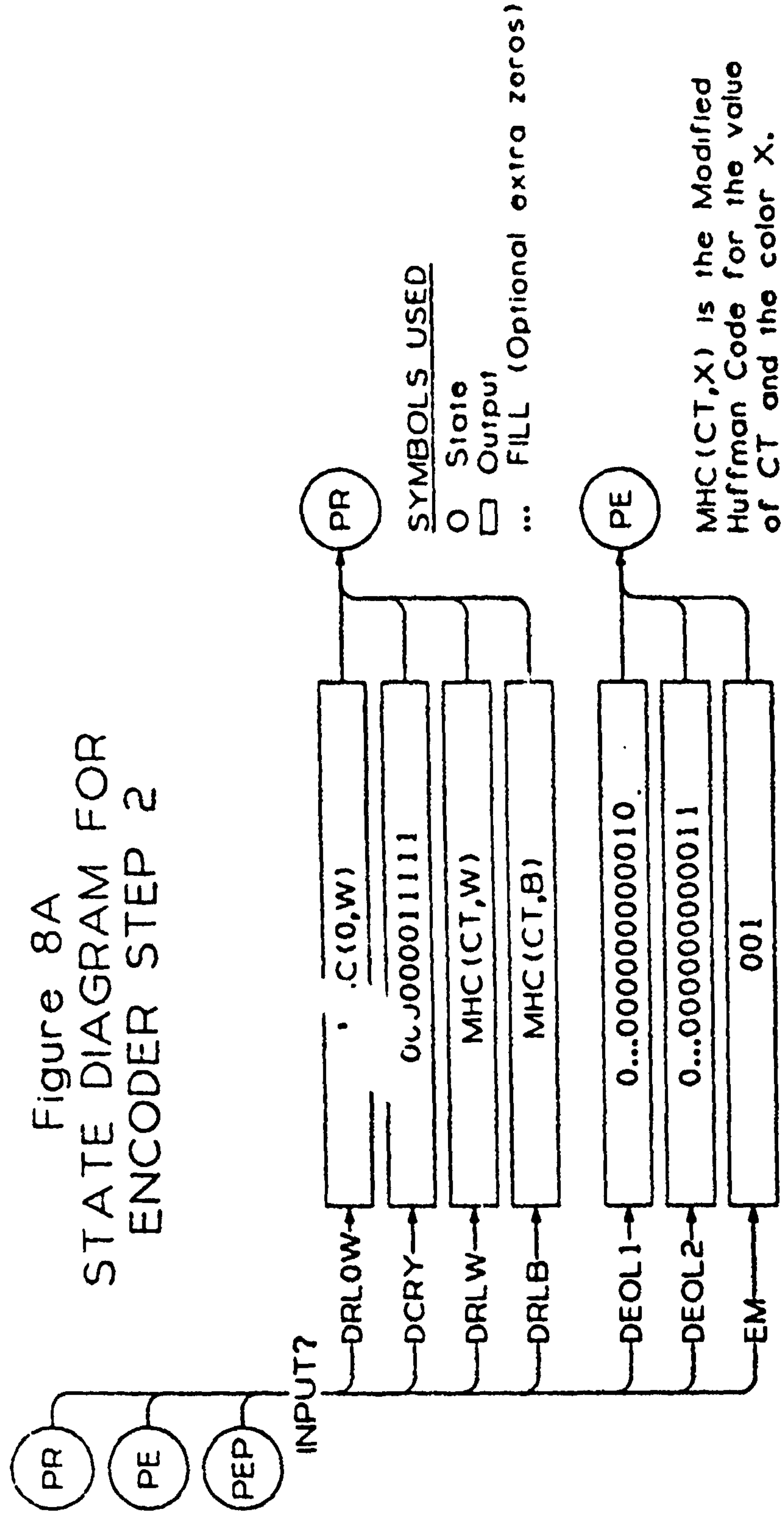
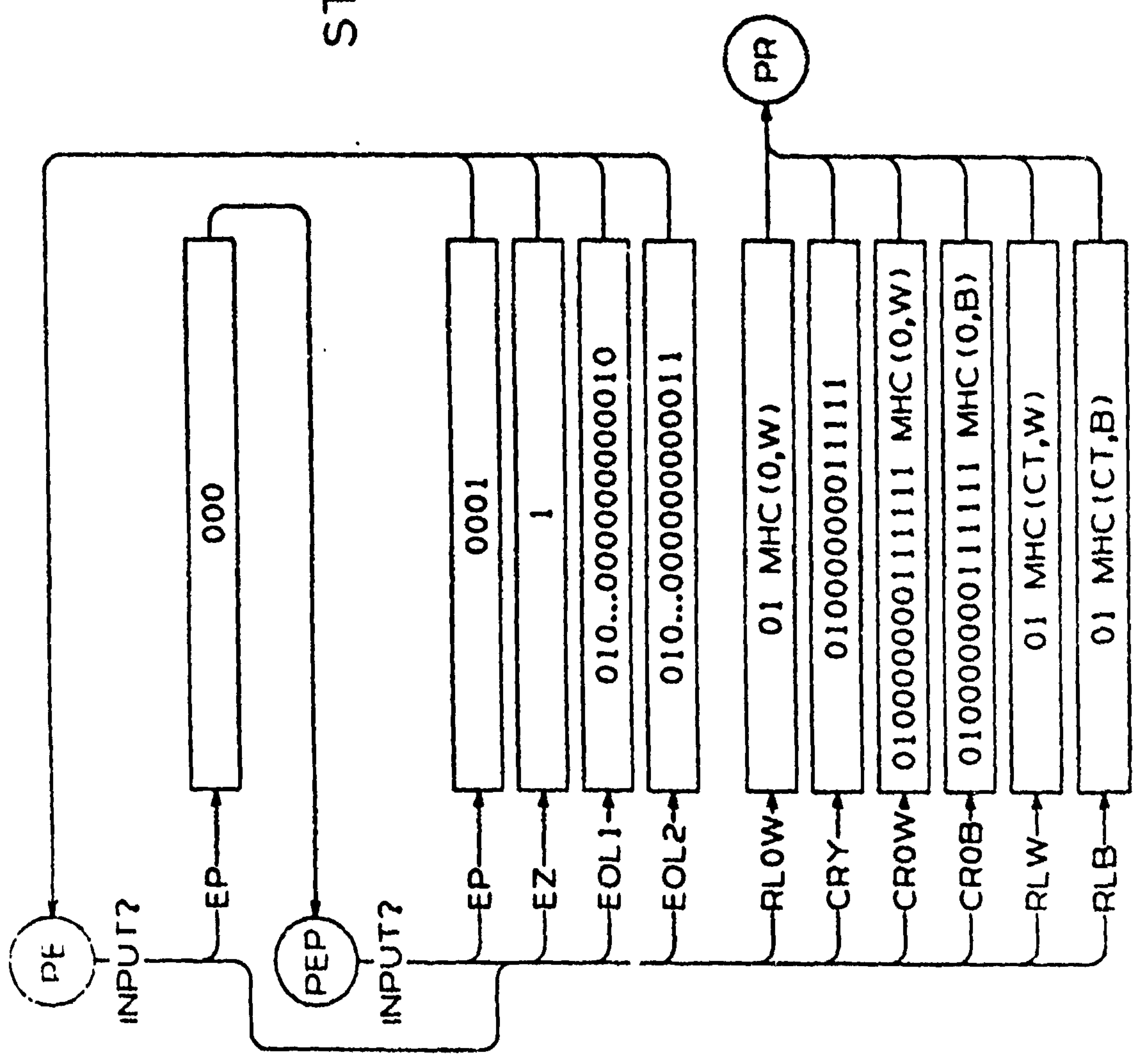
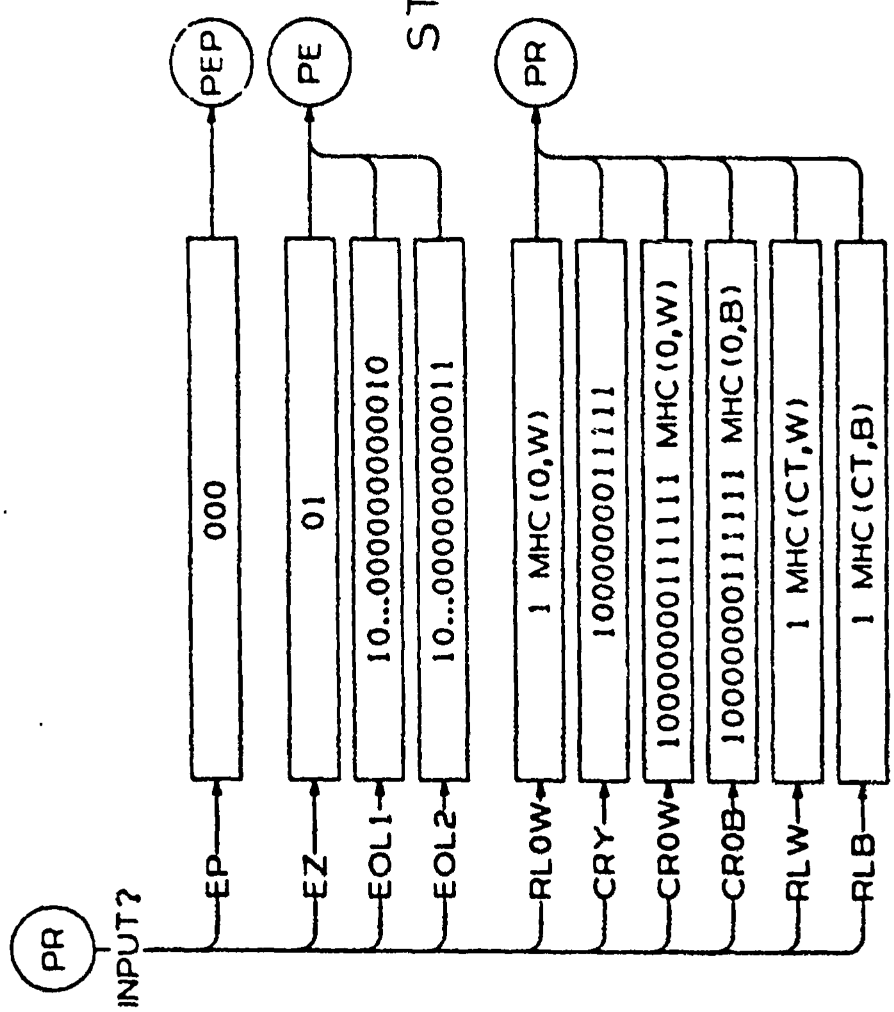


Figure 8B
STATE DIAGRAM FOR
ENCODER STEP 2
(CONTINUED)



(2164)

Figure 8C
STATE DIAGRAM FOR
ENCODER STEP 2
(CONTINUED)



APPENDIX D

CCITT STUDY GROUP XIV

Contribution No. 84

Source: Xerox Corporation

International Telegraph and Telephone
Consultative Committee
(CCITT)

Period 1977-1980

COM XIV-No. [illegible]-E

Original : English

Question : 2/XIV

Date : April 1979

STUDY GROUP XIV - CONTRIBUTION No. 84

SOURCE : XEROX CORPORATION

TITLE : PROPOSAL FOR AN OPTIONAL TWO-DIMENSIONAL CODING SYSTEM FOR GROUP 3 APPARATUS

1. Background

At the November, 1977 meeting of CCITT Study Group XIV, an agreement was reached on all of the standard parameters required for a Group 3 machine. Additionally, a number of optional parameters were defined. These agreements are all recorded in COM XIV-No. 25, pages 33 to 39 in draft Recommendation T.4. Included in Recommendation T.4 is the standard one-dimensional coding scheme to be incorporated in all "standard" Group 3 apparatus. It is further stated in Recommendation T.4 that a two-dimensional coding scheme, offering enhanced operation, may be provided as an option within Group 3 equipment and should be a natural extension of the standard scheme. This two-dimensional coding system is for further study and is the subject of this contribution.

At the December, 1978 meeting of Study Group XIV, a significant amount of discussion centered around the criteria for choosing the two-dimensional optional code. It was finally agreed that the most significant criteria for this selection should include:

- Compression factor
- Error susceptibility
- Complexity and implementation cost
- Patent status

as well as other desirable factors based on the extension of the standard code and the extendability to future codes, resolutions, etc.

Based on a study of those proposals presently contributed to Study Group XIV, Xerox believes that the compression factors and error susceptibility (for a given value of the periodicity of falling back to one-dimensional coding, K) of the codes under consideration are all quite similar. Therefore, strong consideration should be given to such criteria as ease of implementation and patent status.

Although there is no reason to limit the utilization of a two-dimensional coding scheme to those apparatus incorporating other optional features, it should be recognized that the applicability of such coding schemes is best matched to machines utilizing the high definition (7.7 lpmm) and faster scan time options. As this appears to be the prime application, the results of this contribution relate to such a case.

This paper is intended to propose an option for Group 3 equipment. As such, the utilization of this coding scheme in Group 4 apparatus is not stressed. However, one familiar with the concepts agreed to for Group 4 equipment will quickly understand that with minor modifications (such as EOL and K) this technique could easily be used for Group 4 equipment.

2. Discussions

The two-dimensional encoding approach proposed herein is based on a seven element area predictor. This predictor is placed between the source of the video information (scanner) and the run length counter. The predictor examines the picture elements near the pel to be predicted and generates a predicted 1 or 0 (black or white) based on the statistical knowledge of a large number of documents. This predicted value is then compared with the actual pel to determine if the prediction was correct. Correct predictions generate a W symbol and erroneous predictions generate a B symbol. These W and B symbols are then run length encoded by a Huffman approach.

Predictor

A simple block diagram of a one-dimension coder is shown in Figure 1. Figure 2 shows that by selectively inserting a "predictor" block between the data source and the run length counter, the characteristics of the data stream are modified. If the prediction is correct, a "W" is presented at the output of the exclusive OR element and the run length being counted is extended. If an incorrect prediction is made, the Huffman encoder will treat the predicted signal as a change of color.

The predicted value is determined by a statistical analysis of the document data once a particular configuration is chosen. That is, the predicted values are arrived at in essentially the same manner as the choice of the codes in a Huffman look-up table. If, for example, an analysis of documents shows that any particular pixel is nearly always the same color as its immediately preceding pixel, then that would define the prediction algorithm. It can be shown that the percent improvement in compression ratio is nearly linearly related to the improvement in prediction accuracy.

The seven element predictor has been chosen due to its high compression efficiency and simplicity of implementation. It performs within a few percent of predictors containing many more elements over three lines yet require only a fraction of the storage necessary for these more complex configurations. The seven element predictor does offer substantial improvement over a very simple one-element predictor.

Code Table

The coding table structure has been chosen to be identical to that of the standard one-dimensional code table, that is run lengths from 0 to 63 (Terminating Code) are uniquely and individually encoded and multiples of 64 (Make Up Code) are separately encoded. All terminating runs include the "B" element, as the run would only stop if there were a prediction error. Thus, the incorrect predictions are "folded in" to the run length statistics. Only a single code table is required, as the probability of many successive "B's" is extremely low.

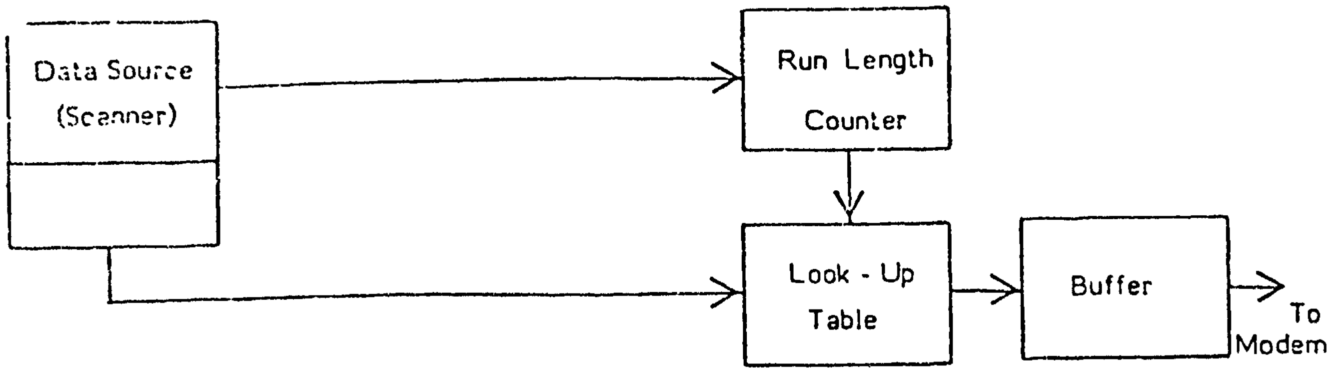


Figure 1 - Block diagram - Huffman coder

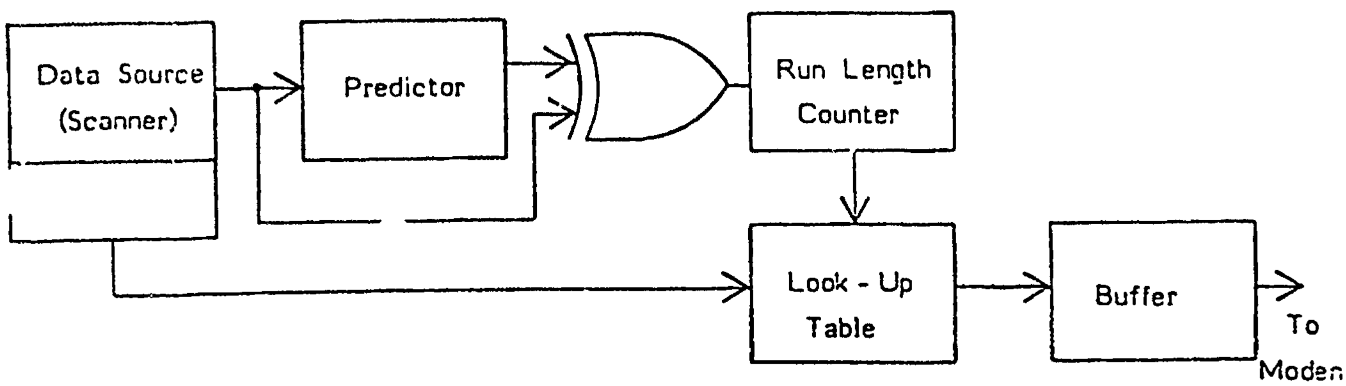


Figure 2 - Block diagram - Huffman coder with predictor

Error Protection

Protection against transmission errors has been provided in several ways. Firstly, as in other proposals submitted to date, a fallback to a one-dimensional approach is recommended every K lines. Any value of K may be chosen with the proposed coding scheme. The results submitted in this contribution use K=4 to maintain consistency with other contributions. The fallback to one-dimensional coding may be accomplished either by using the standard one-dimensional code every K^{th} line or by "zeroing out" the previous reference line of the predictor. The results contained herein utilize the latter approach. Further study is being carried out to determine the relative merits of each method.

As a further consideration for error protection, every pel of the scan line is encoded and a unique end-of-line code is transmitted for each line. This allows the receiving apparatus to fully decode the information and determine if the full 1728 elements have been received. Some gains in compression efficiency could be gained by sending an EOL code immediately following the last B symbol of the line, however, this would negate the possibility of error detections.

A short (5 bit) EOL is used whenever the two-dimensional approach is used and the standard long (11 O's + 1) is used every K^{th} line when the system reverts back to one-dimensional coding. This allows the receiver to completely resynchronize every K^{th} line if errors existed in the transmission. Since the Huffman codes will resynchronize by themselves following an error and also because any error in a line obliterates that line for all practical purposes, full system forced resynchronization more often than every K^{th} line is deemed unnecessary.

FILL

As is the case with the one-dimensional standard, a need exists to be able to insert a FILL code whenever devices built to different specifications are interoperating. This proposal recommends inserting such FILL code every K^{th} line when the long EOL is used. Thus, if K=4 and the T.30 protocol signals a 5 millisecond per line capability, this can be interpreted at the sender as 4x5=20 milliseconds every K lines. If the compressed data plus the short EOL codes use less than this, FILL will be added at the beginning of the long EOL, preceding the one-dimensionally encoded line of data.

Patent Status

To date, a thorough patent search has not been completed to determine if the material contained in this proposal is covered by any existing patents. However, this material is believed to be free of any encumbering patents which would limit its availability. Xerox believes in the importance of the universal availability of the technologies standardized by C.C.I.T.T. and will pursue, via a patent search, the answer to this question.

Performance

A full discussion of the performance of the approach proposed herein will not be given in this contribution. It will be the subject of a companion contribution to be sumbitted in the near future. However, in order for the readers to gain a general understanding of the compression factor which is capable of being achieved by this approach the following data is presented.

The average compression of all eight C.C.I.T.T. test documents at 1728 pels per line and 7.7 lines per millimeter is 11.57. This data is for a system with K=4 and includes EOL codes on each line. No FILL has been inserted. At a 4800 bps transmission rate, this yields an average transmission time of 66 seconds, when operating in the optional high resolution mode. Compression factors (times) for the eight documents range from 5.5 (140 seconds) to 24 (30 seconds).

3. Proposal

The structure of the proposed predictor is shown in Figure 3. The prediction table is exhibited in Table 1. The proposed Hoffman code table to run length encoded the W or B output of the predictor is given in Table 2.

X_0	X_1	X_2	X_3	X_4
X_5	X_6	Y		

Figure 3 - Seven element predictor

TABLE 1

Proposed prediction table

NO	X0	X1	X2	X3	X4	X5	X6	Y
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1
2	0	0	0	0	0	1	0	0
3	0	0	0	0	0	1	1	1
4	0	0	0	0	1	0	0	0
5	0	0	0	0	1	0	1	1
6	0	0	0	0	1	1	0	0
7	0	0	0	0	1	1	1	1
8	0	0	0	1	0	0	0	0
9	0	0	0	1	0	0	1	1
10	0	0	0	1	0	1	0	0
11	0	0	0	1	0	1	1	1
12	0	0	0	1	1	0	0	0
13	0	0	0	1	1	0	1	1
14	0	0	0	1	1	1	0	0
15	0	0	0	1	1	1	1	1
16	0	0	1	0	0	0	0	0
17	0	0	1	0	0	0	1	1
18	0	0	1	0	0	1	0	0
19	0	0	1	0	0	1	1	1
20	0	0	1	0	1	0	0	0
21	0	0	1	0	1	0	1	1
22	0	0	1	0	1	1	0	1
23	0	0	1	0	1	1	1	1
24	0	0	1	1	0	0	0	1
25	0	0	1	1	0	0	1	1
26	0	0	1	1	0	1	0	1
27	0	0	1	1	0	1	1	1
28	0	0	1	1	1	0	0	1
29	0	0	1	1	1	0	1	1
30	0	0	1	1	1	1	0	1
31	0	0	1	1	1	1	1	1
32	0	1	0	0	0	0	0	0
33	0	1	0	0	0	0	1	1
34	0	1	0	0	0	1	0	0
35	0	1	0	0	0	1	1	1
36	0	1	0	0	1	0	0	0
37	0	1	0	0	1	0	1	1
38	0	1	0	0	1	1	0	1
39	0	1	0	0	1	1	1	1
40	0	1	0	1	0	0	0	0
41	0	1	0	1	0	0	1	1
42	0	1	0	1	0	1	0	0
43	0	1	0	1	0	1	1	1
44	0	1	0	1	1	0	0	0
45	0	1	0	1	1	0	1	0
46	0	1	0	1	1	1	0	0
47	0	1	0	1	1	1	1	1
48	0	1	1	0	0	0	0	0
49	0	1	1	0	0	0	1	1
50	0	1	1	0	0	1	0	0
51	0	1	1	0	0	1	1	1
52	0	1	1	0	1	0	0	0
53	0	1	1	0	1	0	1	1
54	0	1	1	0	1	1	0	0
55	0	1	1	0	1	1	1	1
56	0	1	1	1	0	0	0	1
57	0	1	1	1	0	0	1	1
58	0	1	1	1	0	1	0	0
59	0	1	1	1	0	1	1	1
60	0	1	1	1	1	0	0	1
61	0	1	1	1	1	0	1	1
62	0	1	1	1	1	1	0	0
63	0	1	1	1	1	1	1	1
64	1	0	0	0	0	0	0	0
65	1	0	0	0	0	0	1	1
66	1	0	0	0	0	1	0	0
67	1	0	0	0	0	1	1	0
68	1	0	0	0	1	0	0	0
69	1	0	0	0	1	0	1	1
70	1	0	0	0	1	1	0	0
71	1	0	0	0	1	1	1	1
72	1	0	0	1	0	0	0	0
73	1	0	0	1	0	0	1	1
74	1	0	0	1	0	1	0	0
75	1	0	0	1	0	1	1	1
76	1	0	0	1	1	0	0	0
77	1	0	0	1	1	0	1	1
78	1	0	0	1	1	1	0	0
79	1	0	0	1	1	1	1	1
80	1	0	1	0	0	0	0	0
81	1	0	1	0	0	0	1	1
82	1	0	1	0	0	1	0	0
83	1	0	1	0	0	1	1	1
84	1	0	1	0	1	0	0	0
85	1	0	1	0	1	0	1	1
86	1	0	1	0	1	1	0	1
87	1	0	1	0	1	1	1	1
88	1	0	1	1	0	0	0	0
89	1	0	1	1	0	0	1	1
90	1	0	1	1	0	1	0	1
91	1	0	1	1	0	1	1	1
92	1	0	1	1	1	0	0	0
93	1	0	1	1	1	0	1	1
94	1	0	1	1	1	1	0	1
95	1	0	1	1	1	1	1	1
96	1	1	0	0	0	0	0	0
97	1	1	0	0	0	0	1	1
98	1	1	0	0	0	1	0	0
99	1	1	0	0	0	1	1	0
100	1	1	0	0	1	0	0	0
101	1	1	0	0	1	0	1	1
102	1	1	0	0	1	1	0	0
103	1	1	0	0	1	1	1	0
104	1	1	0	1	0	0	0	0
105	1	1	0	1	0	0	1	1
106	1	1	0	1	0	1	0	0
107	1	1	0	1	0	1	1	1
108	1	1	0	1	1	0	0	0
109	1	1	0	1	1	0	1	1
110	1	1	0	1	1	1	0	0
111	1	1	0	1	1	1	1	1
112	1	1	1	0	0	0	0	0
113	1	1	1	0	0	0	1	1
114	1	1	1	0	0	1	0	0
115	1	1	1	0	0	1	1	1
116	1	1	1	0	1	0	0	0
117	1	1	1	0	1	0	1	1
118	1	1	1	0	1	1	0	0
119	1	1	1	0	1	1	1	1
120	1	1	1	1	0	0	0	0
121	1	1	1	1	0	0	1	1
122	1	1	1	1	0	1	0	0
123	1	1	1	1	0	1	1	1
124	1	1	1	1	1	0	0	0
125	1	1	1	1	1	0	1	1
126	1	1	1	1	1	1	0	0
127	1	1	1	1	1	1	1	1

TABLE 2

RUN	CODE LENGTH	HUFFMAN CODE
W 0+B	3	111
W 1+B	4	1011
W 2+B	3	110
W 3+B	4	1010
W 4+B	4	1001
W 5+B	4	1000
W 6+B	5	01111
W 7+B	5	01110
W 8+B	5	01101
W 9+B	6	010101
W10+B	6	010100
W11+B	6	010011
W12+B	6	010010
W13+B	7	0100001
W14+B	7	0100000
W15+B	7	0011111
W16+B	7	0011110
W17+B	7	0011101
W18+B	7	0011100
W19+B	7	0011011
W20+B	7	0011010
W21+B	7	0011001
W22+B	7	0011000
W23+B	7	0010111
W24+B	7	0010110
W25+B	7	0010101
W26+B	7	0010100
W27+B	7	0010011
W28+B	7	0010010
W29+B	8	00011001
W30+B	8	00011000
W31+B	8	00010111
W32+B	8	00010110
W33+B	7	0010001
W34+B	8	00010101
W35+B	7	0010000
W36+B	7	0001111
W37+B	8	00010100
W38+B	8	00010011
W39+B	8	00010010
W40+B	8	00010001
W41+B	8	00010000
W42+B	8	00001111
W43+B	8	00001110
W44+B	9	000010111
W45+B	9	000010110
W46+B	9	000010101
[illegible]	[illegible]	[illegible]

RUN	CODE LENGTH	HUFFMAN CODE
W48+B	9	000010011
W49+B	8	00001101
W50+B	9	000010010
W51+B	9	000010001
W52+B	9	000010000
W53+B	9	000001111
W54+B	9	000001110
W55+B	9	000001101
W56+B	9	000001100
W57+B	9	000001011
W58+B	9	000001010
W59+B	9	000001001
W60+B	9	000001000
W61+B	9	000000111
W62+B	9	000000110
W63+B	9	000000101
W 64	5	01100
W 128	6	010001
W 192	7	0001110
W 256	8	00001100
W 320	9	000000100
W 384	9	000000011
W 448	12	000000010001
W 512	12	000000010000
W 576	12	000000001111
W 640	12	000000001110
W 704	12	000000001101
W 765	12	000000001100
W 832	12	000000001011
W 896	12	000000001010
W 960	12	000000001001
W1024	12	000000001000
W1088	12	000000000111
W1152	12	000000000110
W1216	12	000000000101
W1280	12	000000000100
W1344	12	000000000011
W1498	12	000000000010
W1472	12	000000010011
W1536	12	000000010010
W1600	11	00000001011
W1664	11	00000001010
W1728	7	0001101
EOL1	5	01011
EOL	12	000000000001

4. Summary

The predictor and encoding system proposed in this document is felt to meet all of the criteria set forth by C.C.I.T.T Study Group XIV at its December 1978 meeting. Xerox Corporation recommends that this easy-to-implement and effective coding method be adopted by Study Group XIV as the optional two-dimensional scheme and be incorporated into Recommendation T.4.

APPENDIX E

CCITT STUDY GROUP XIV

Contribution No. 81

Source: AT&T

Consultative Committee
(CCITT)

Period 1977-1980

Original : English

Question : 2/XIV

Date : March 1979

STUDY GROUP XIV - CONTRIBUTION No. 81

SOURCE : AT & T

TITLE : PROPOSAL FOR TWO-DIMENSIONAL FACSIMILE CODING SCHEME

CONTENTS

1. INTRODUCTION

Working Party 2 at the Study Group XIV meeting of December 11 to 15, 1978 set forth a time table for submission, as well as uniform selection criteria for evaluation of two-dimensional facsimile coding schemes proposed for Group 3 machines. It is widely recognized that a two-dimensional code is necessary to improve the compression performance of a Group 3 machine. In addition, a two-dimensional scheme will offer increased coding efficiencies in new classes of terminals which operate in error-free code environments, where the two-dimensional algorithm can be applied throughout the entire image without restart.

The proposed algorithm, termed Frank code [1], is a simple extension of the one-dimensional Modified Huffman Code. This code gives superior coding efficiency when operated either in a Group 3 terminal or in an error-free environment. It has the advantage that it is based on statistically relevant image features. There is strong indication that new, more efficient feature-based algorithms can be designed in the future using this code as their base. In addition, there are simple ways to extend this code linguistically to incorporate new functions such as multiplexing non-graphics data into the code stream, or coding gray scale or color images.

The Frank coding algorithm may be implemented simply in a wide variety of ways. It can be segmented into several independent processes which can lead to efficient system designs. This code has been implemented in a variety of systems and has been found to be a highly efficient and reliable algorithm. Based on these factors and others presented in the body of this proposal, AT&T recommends that Frank code be adopted as a standard two-dimensional code for use in facsimile terminals.

2. TWO-DIMENSIONAL CODING SCHEME

2.1 OVERVIEW OF CODING SCHEME

Frank coding is a two-dimensional, feature-oriented, statistically-based coding scheme with high coding efficiency, small memory requirements, and simple logic. It is a form of line-to-line differential run length coding, with the added distinction that it identifies and codes "blobs", which are collections of geometrically associated picture

elements (pels) of like brightness level. We assume black to be the level encoded.

In brief, the coding structure is as follows. A is a set of black runs on successive image lines such that two conditions are satisfied. First, a blob may contain no more than one run per image line. Second, given any two runs on successive lines in a blob, their left ends are no more than three pels apart, and their right ends are no more than three pels apart. The parameters coded for a blob are (a) the position of the blob within the image, (b) the length of the topmost or head run, (c) the way in which the left ends and the right ends connect for each two successive runs in a blob, and (d) in some cases an indication of the final or end run coded. The CCITT standard one-dimensional white-run-length Modified Huffman codebook is used to code the blob position parameter, simultaneously providing downward compatibility with earlier one-dimensional code machines. A separate codebook includes entries for 93 blob parameters.

Behavior of the blob parameters has proved to be statistically very stable over a large ensemble of images. This consistency results from the close relation of the blobs to micro-structures within an image which form a common basis of facsimile documents. This relation is keyed to the restriction that two runs on successive lines in a blob may be displaced from each other no more than three pels on either side. In effect this controls blob contours and captures two-dimensional coherence which is reflected in high coding efficiency. A more extensive discussion of the particular choice of the constraint of three is given in Appendix A.

The blob organization of data gives greater control over defining approximate encodings, which simultaneously increase coding efficiency and maintain high fidelity to the original image. Also, there is strong indication that the blob data may be used for pattern recognition purposes to identify vectors, characters, or higher level image structures. As one example, Pferd and Ramachandran [2] discuss a computer-aided automatic digitizing system for encoding, editing, and displaying engineering documents by converting Frank code to vector representation. On a test document Frank coding yielded a compression factor of 20, and the vector representation increased this to 36. It is to be noted that both the blob parameters and the vectors were coded in raw form, and that efficient coding would result in a higher compression factors. We also note that the indicated vector representation gives a high fidelity approximation to the original image. In addition to improving compression, the vector representation permits changes to the image to be made easily either automatically or by operator intervention. The authors also discuss the extension of the system to recognize characters and "elements"

such as rectangles. With this extension the system is expected to result in a compression ratio of 180. The use of blob data to encode image features is also discussed in section 4 below.

2.2 DETAILED DESCRIPTION OF CODING SCHEME

2.2.1 Basic Coding Rules

We encode by comparing the black runs on two sequential scan lines, called line A and line B. In general terms, if a line B run is displaced from a line A run by no more than three pels on both the left and right sides separately, then a CONNECT code may be issued. As discussed in Appendix A, our studies have shown that a displacement of three results in both coding efficiency and in definition of blob structures which are of meaningful size and coherence for feature-based operations. If a line B run does not connect to any line A run, then the line B run starts a new blob, and a HEAD code is issued. Following the initial start or any restart of the encoding algorithm, in the first image line which contains any black runs, all the runs are heads of blobs. If a line A run does not connect to any line B run, then the line A run is the last run in a blob, and an END code is issued. In some cases, END codes are redundant and are accordingly deleted as indicated further below.

More specifically, assume a line scanned image and focus attention at the left margin. Consider the two sequential scan lines called line A and line B. The relative positions of the white-to-black and the black-to-white transitions in these two lines are compared, and codes issued in accordance with five basic coding rules. For illustration purposes, we present these rules in terms of pointer mechanisms which search the two lines for the transitions. Many different implementations using parallel or serial access or a combination thereof, can be realized. In this discussion, assume one pointer which moves along line A, and another pointer which moves along line B, from left to right. Note the scan coordinates of the first white-to-black transition and the first black-to-white transition in each line. In line A, call the coordinate of the first black pel A1. In line B, call the coordinate of the first black pel B1. Call the coordinates for the first white pel after the first black-to-white transition, A2 and B2, for the two lines respectively. The {A1,B1} and {A2,B2} sets define two black runs, one in each line. We compare the two black runs indicated by these sets and encode according to the following rules, which are illustrated in Figure 1:

1) If $A1-B1 > 3$, issue a HEAD code, indicating that the run in line B is the first run in a blob, and advance the line B pointer to the next white-to-black and black-to-white transitions, B1 and B2.

2) If $|A1-B1| \leq 3$ and $A2-B2 > 3$, issue a HEAD code for the run in line B, and advance the line B pointer to the next B1 and B2.

3) If $|A1-B1| \leq 3$ and $|A2-B2| \leq 3$, issue a CONNECT code, indicating that the runs in lines A and B connect to each other in the same blob, and advance both line A and line B pointers to the next (A1,B1) and (A2,B2).

4) If $B1-A1 > 3$, issue an END code, indicating that the run in line A is the last run in a blob, and advance the line A pointer to the next white-to-black and black-to-white transitions, A1 and A2.

5) If $|A1-B1| \leq 3$ an $B2-A2 > 3$, issue an END code for the run in line A, and advance the line A pointer to the next A1 and A2.

Upon advancing the line A pointer, if the end of the line is reached before a white-to-black transition, then rule (1) holds for any remaining black runs in line B. This condition also applies for the first line in the image, and for the first line any time after the algorithm is restarted. Thus, all black runs in these lines are HEADS. Upon advancing the line B pointer, if the end of the line is reached before a white-to-black transition, then rule (4) holds for any remaining black runs in line A.

2.2.2 Blob Parameter Codes

In the following we describe the codeword structure for the HEAD, CONNECT, and END codes. A HEAD code contains two pieces of information, effectively giving the length of the first run in the blob, and the position of the blob in the image. For the position information we use the standard one-dimensional white-run-length Modified Huffman codewords. For the other blob parameters we use a newly constructed Huffman codebook. This new codebook is completely separate from the standard one-dimensional white-run-length Modified Huffman codebook. The separation is possible because the head position code always follows the code for the head length. The two codebooks may in fact contain the same values, which of course have different meanings depending upon the codebook accessed. Appendix B shows a specific blob parameter codebook along with the frequency data of the

blob parameters for the ensemble of the eight standard CCITT test images. This data assumes digitization at full resolution and a K factor of infinity, as discussed further below. The blob parameter Huffman codes are based on this frequency data, with adjustments to accomodate the end of line (EOL) code, the fill bits which may be necessary to attain a minimum transmission time per image line, and resynchronization procedures, as discussed in full further below. We now discuss the blob parameter codeword structure in detail.

1) HEAD code, position

The blob position is the number of white pels displacement, D, of the current black head run from either (a) the previous black run, or, (b) the left hand margin if the run is the first black run in an image line. This last condition is applicable because in a Group 3 machine each line is terminated with an EOL code. This parameter is coded with the standard one-dimensional white-run-length Modified Huffman codewords.

2) HEAD code, length

The length, L, of the head run is given by a Huffman codeword, or, in the case of "long" heads, by a Huffman codeword followed by a suffix code. Those runs 40 pels or less are assigned unique Huffman codewords. Runs exceeding 40 pels are classified into three categories. The first category contains the 31 run lengths of 41 through 71. The second category contains the 511 run lengths of 72 through 582, and the third category contains all run lengths exceeding 582. Each of the three categories is assigned a unique Huffman codeword. The code issued for a head in the first category consists of the Huffman codeword for the first category, and 5 additional bits to indicate the particular member in the category. Specifically the 5 bits comprise the binary value of the head length minus 40. Similarly, the code for a head in the second category consists of the Huffman codeword for the category, and 9 bits which contain the binary value of the head length minus 71. Finally, the code for a head in the third category consists of the Huffman codeword for the category, and 11 bits which contain the binary value of the head length minus 582.

Resynchronization is discussed in detail further below. At this point we note that in no case does the 5, 9, or 11 suffix consist of all zeroes. Also for resynchronization purposes, if any of the 5, 9, or 11 suffix bits end in four zeroes, then an additional 1 is appended. In these cases, the suffix code consists of 6, 10, and 12 bits for the three long head categories respectively. This condition occurs rarely. Firstly,

it occurs for less than 1 out of each 16 binary values of the suffix bits. Secondly, as can be seen from the frequency data in Appendix B, the number of long heads, i.e. those of length 41 through 2560 inclusive, is a total of 1076, as compared for example to the number of heads of length three or less, which is a total of 33,307.

3) CONNECT code

A CONNECT code defines both the left hand displacement, LHS = B1-A1, and the right hand displacement, RHS = B2-A2. As there are seven left hand displacements and seven right hand displacements, there are 49 different Huffman CONNECT codes.

4) END code

An END code is the same for all blobs. Therefore there is only one Huffman END code.

In summary, there are 43 HEAD length codes, 49 CONNECT codes, and 1 END code, yielding a total of 93 blob parameter codes. The coding for one blob consists of a HEAD code and one CONNECT code for each run in the blob on scan lines following the head run, and in some cases an END code. In the encoded bit stream we freely intermix the HEAD, CONNECT, and END codes for one blob with the same codes for other blobs. This is illustrated in the example shown in Figure 2.

It should be noted that an EOL code is not required for correct decoding. Thus, for example, consider the decoding of the codes indicated for the example in Figure 2. As in the encoding process we may visualize the decoding process as employing two pointers, one for the last line constructed, called line A, and one for the next line to be constructed, called line B. To construct line 1, we may use a fictitious line, 0, as line A, and line 1 as line B. We assume line 0 has no black runs in it. The first code, HEAD(L=4,D=0), causes the run J1 to be constructed. The next code, CONNECT(LHS=2,RHS=0), causes the decoder to move the line A pointer, searching for the next black run in line 0. The search starts at the left end of line 0, and reaches the end of the line without having found a black run. When this happens the decoder switches to the next two lines in sequence, i.e. line 1 now becomes line A, and line 2 becomes line B. The search for a black run now continues in the new line A, and the run J1 is immediately found. The CONNECT(LHS=2,RHS=0) is made relative to the J1 run, thereby constructing run J2. The next code, HEAD(L=2,D=5), causes the head run, K1 to be constructed. The next code, CONNECT(LHS=-1,RHS=-1), again causes the decoder to move the line A pointer, searching for the next black run in line 1.

(2595)

The search starts to the right of run J1, and again reaches the end of the line without having found a black run. The decoder switches to the next two lines in sequence, i.e. line 2 now becomes line A, and line 3 becomes line B. The search for a black run now continues in the new line A, and the run J2 is found. The CONNECT(LHS=-1,RHS=-1) is made relative to the J2 run, thereby constructing run J3.

Thus we see how CONNECT codes may be properly decoded with no reference to any EOL codes. This is also the case for HEAD codes. However, if EOL codes are inserted because of the standard Group 3 requirement, then advantage is taken of them by specifying the displacement, D, of the first black run in any image line to be measured relative to the left hand margin, as indicated for example by the head run M1 in Figure 2.

In addition, in an error free environment or one permitting retransmission, EOL codes are not required. Then we may consider the bit stream representing the image to be continuous, and the end of an image line is "wrapped around" to continue immediately with the beginning of the next line. In this case, the displacement, D, measures the number of white pels between the current black head run and the previous black run. These two runs may not be on the same image line. For example, the displacement for the head run, N1, in Figure 2, would be 4, because there are 4 white pels between run N1 and the previous black run, K5. The two runs may in fact be separated by any number of lines. The displacement, D, may exceed the maximum value of 2560. To code a displacement equal to or greater than 2560, we divide the displacement by 2560, giving an integral quotient, Q, and a remainder, R, which is less than 2560. The codeword corresponding to the run length value of 2560 is issued Q times, followed by the codeword for the remainder, R. For R equal to zero, the codeword for run length zero is issued.

2.2.3 Deletion of Redundant END Codes

END codes serve as "tab commands" for a decoder. For example consider the codes shown in Figure 2. Assume that line 3 is already decoded. The next three codes in the bit stream are the two ENDS and the CONNECT(LHS=-1,RHS=1). The first END code causes the decoder to tab beyond the first black run, J3, in line 3, and the second END code causes a tab beyond the next black run, I1, in line 3. The CONNECT is then made properly to the next run, K2, in line 3, resulting in the construction of run K3. If the END codes were absent, then the CONNECT would have been made erroneously to run J3. In some cases the END codes issued according to the five rules given previously are redundant.

In Frank code, three redundancies are recognized. These are termed the HEC (HEAD, END, CONNECT) type, the HEH (HEAD, END, HEAD) type, and the CEH (CONNECT, END, HEAD) type. In each case, the E represents one or a number of END codes.

The HEC redundancy is illustrated in Figure 3. The redundancy can be seen by decoding the blob data. Assume that line A is already constructed, and that line B remains to be reconstructed. The first code, HEAD(L=12,D=0), is constructed directly. Then the END codes for blobs G, H, I, J, and K, simply "tab" across line A so that the CONNECT(LHS=-2,RHS=0) is made to run L1, resulting in the construction of run L2. However, the END codes for blobs G, H, and I, can be deleted. These runs can not possibly connect to any black run succeeding the head run M, because the left sides of runs G, H, and I are more than three pels away from any black run succeeding run M. Specifically, the left sides of G, H, and I are at or before pel position 10, and the left side of any run succeeding run M must be at or after pel position 14. By the same token, the END codes for blobs J and K must be retained. As indicated in the figure, any END run starting prior to pel position 11 may be deleted, and any END run starting at or after pel position 11 must be retained. The value of B2 for run M is 13. Thus, in general, any run in line A such that B2-A1 > 2 may be deleted.

A coded bit stream in which END codes have been deleted requires a simple change to the decoding process. Specifically, when a HEAD is decoded, leaving the line B pointer at the position B2, then the line A pointer is moved to position B2-3. The codes for any END runs starting at or to the left of this position in line A were deleted. The actions taken at this point depend upon the next code in the bit stream. If this code is another HEAD code, then the head run is constructed, and the line A pointer is advanced to the new position B2-3. If this code is an END or a CONNECT code, then the decoder searches for the first white-to-black transition in line A to identify the next occurring black run. The earliest position at which such a black run may start is B2-2. In the example shown in Figure 3, assume that the END codes for runs G, H, and I have been deleted and that the END codes for runs J and K have been retained. The HEAD(L=12,D=0) is decoded and run M constructed, leaving B2 at pel position 13. Next, the line A pointer is advanced to position B2-3, which is position 10. The next code in the bit stream is the first END code. In this case, the decoder starts at position 10 and searches for the first white-to-black transition. This occurs between pel position 11 and 12. Therefore, run J is identified as the next occurring black run. The END code is applied to this run, and the decoder "tabs" beyond it. Similarly, the second END code is applied to run K, and the decoder likewise "tabs"

beyond it. Now only the CONNECT(LHS=-2,RHS=0) code remains to be processed. The decoder finds run L1 as the next run in line A, applies the CONNECT to it, and constructs run L2.

The HEH redundancy is illustrated in Figure 4. Again the redundancy can be seen by decoding the blob data. Assume that line A is already constructed and that line B remains to be reconstructed. The first head code, HEAD(L=6,D=0), is reconstructed relative to the left margin. The second head code, HEAD(L=3,D=3) is reconstructed relative to the right end of the first head. To construct these heads, no reference need be made at all to the two END codes which terminate blobs I and J. Furthermore, any END code which occurs between the two heads, I and M, is for a run which under no condition can connect to a run which succeeds run M in line B. Consider for example run J. In order for it to be an END preceding the HEAD for run M, it must satisfy rule (4) or rule (5). Either A1 for run J must be more than three to the left of B1 for run M, or, A2 for run J must be more than three to the left of B2 for run M. In either case, A1 for run J is more than three to the left of B1 for run K2, making a connection impossible. Thus, any END run occurring between two HEADS is not needed for constructing the HEAD runs or any succeeding CONNECT runs.

If these ENDS are deleted, they can be bypassed easily during the decoding process in the same manner as indicated before. It is to be noted that a run in line A starting at pel position 9 or 10, can not be an END run preceding the HEAD M. To begin with, such runs do not satisfy rule (4), i.e. their left sides are within a distance of three from the left side of run M. Also, such runs do not satisfy rule (5). In particular, A1 for these runs is not more than three to the left of B2 for run M. This means that A2 for these runs is also not more than three to the left of B2 for run M, thus violating rule (5). Since these runs can satisfy neither rule (4) nor rule (5), they can not be ENDS preceding the HEAD M. However, they are ENDS following the HEAD M, and preceding the CONNECT for run K2. They will be deleted under the HEC redundancy type deletion process described previously, because in each case, B2-A1 > 2. Thus, the decoder may proceed in this case as in the previous example, bypassing all runs in line A with B2-A1 > 2.

The CEH redundancy is similar to the HEH redundancy and may be treated the same way in the decoding process.

We have illustrated the deletion of redundant END codes by explaining how to handle them in the decoding process. In the encoding process, redundant END codes can be deleted by the following rules.

1) Upon identification of an END condition, the END code is not issued immediately, but actions are taken depending upon preceding conditions:

   a) If the preceding conditions were a CONNECT followed by any number of ENDS, (CEE...E), then a specified "end" counter is increased by 1 for the current END condition. In this case, the end counter contains a count of the number of successive ENDS that have been identified following a CONNECT code.

   b) If the preceding conditions were a HEAD followed by any number of ends, (HEE...E), then the end counter is increased by 1 for the current END condition only if A1+2 > B2, where A1 is the coordinate of the first pel in the run identified as an END, and B2 is the coordinate of the pel following the last pel in the preceding HEAD run. In this case, the end counter contains a count of the number of successive ENDS that have been identified following a HEAD code, other than those ENDS which can not possibly connect to any black runs succeeding the indicated HEAD.

2) Upon identification of a HEAD condition, no END codes are issued, the end counter is reset to 0, and then the HEAD code is issued.

3) Upon identification of a CONNECT condition, as many END codes are issued as indicated in the end counter, the counter is reset to 0, and then the CONNECT code is issued.

2.2.4 End of Line, Fill, and Return to Control Codes

The EOL code is eleven zeroes followed by a 1. Fill bits of zeroes, where required to attain a specified minimum transmission time per image line, are inserted before the EOL code. To distinguish the EOL code from the blob parameter codes, we have prohibited the blob parameter codes from having a prefix of eleven zeroes. As can be seen from the blob parameter codebook in Appendix B, no blob parameter codeword starts with eleven zeroes, and thus the configuration for the EOL code, or for the EOL code with fill bits, can be uniquely determined. Upon decoding such a string of eleven zeroes, the decoder searches the following bit stream for the first occurring 1. Any zeroes encountered in this search are fill bits. To provide for the EOL code, or the EOL code with fill bits, requires that a few blob parameter codes have one more bit than if this provision did not have

to be made. It should be noted that many realizations of a Huffman code for a given frequency distribution are possible. The realization shown in Appendix B is highly efficient in that the few blob parameter codes extended by one bit are those with the most infrequent occurrences. The codebook also meets resynchronization requirements as indicated further below.

The return to control code consists of the standard six EOL codes.

2.2.5 Blob Parameter Codebook with Supplementary Codes

Appendix B shows the codebook containing the 93 blob parameter codes. Observe that the rules outlined above form the primitive elements or words of a language which describes two-dimensional run length correlation. As the image is encoded, it is translated from the binary pel-by-pel language to the language of HEADS, CONNECTS, and ENDS. A particularly attractive implementation feature is to permit the future extension of this language to include other primitives. Such primitives may, for example, code gray scale and color, or they may relate to extended image operations, or they may be codes to control the receiver in various ways. Economies can be realized by using the same mechanism to decode these functions as that used to identify the basic language primitives. We therefore urge that the blob parameter codebook be augmented with 12 supplementary unassigned codes, as shown in Appendix C. If the expanded codebook in Appendix C is used rather than the codebook in Appendix B, then only 158 additional bits are required to code all eight test images, or, less than 20 additional bits per image on the average. This assumes digitization at full resolution, and a K factor of infinity, as described further below.

2.2.6 Resynchronization

If transmission conditions result in loss of synchronization, so that the beginning of a codeword becomes unknown, the decoder may resynchronize by searching for a pattern of eleven zeroes and a 1. This is the EOL code, or, the terminal portion of an EOL code with fill bits. This pattern can not be generated from any combination of codewords. To insure this condition the blob parameter codes obey the following rules.

(2595)

1) No HEAD, END, or CONNECT code consists of all zeroes or contains any initial, terminal, or internal segment consisting of eleven zeroes.

2) No succession of any two HEAD, END, or CONNECT codes can result in eleven zeroes. As seen from Appendix B, the maximum number of initial zeroes of any of these codes is five, and the maximum number of terminal zeroes is four.

3) No succession of any HEAD code for the head lengths of 1 through 40, followed by any one-dimensional white-run-length Modified Huffman code can result in eleven zeroes. As can be seen from Appendix B, the maximum number of terminal zeroes for any HEAD code is three, and the maximum number of initial zeroes for the run length codes is seven. We note that the maximum number of terminal zeroes for any CONNECT code is four, but that a CONNECT code is never followed by a run length code.

4) No succession of any one-dimensional white-run-length Modified Huffman code, followed by any HEAD, END, or CONNECT code can result in eleven zeroes. This follows because the maximum number of terminal zeroes in the run length code is three, and the maximum number of initial zeroes in any blob parameter code is five.

5) No succession of any "long head" category code followed by an associated suffix code can result in eleven zeroes because each "long head" category code ends in a 1, and no "long head" suffix code consists of all zeroes.

6) No succession of any "long head" suffix code followed by any one-dimensional white-run-length Modified Huffman code can result in eleven zeroes. This follows because any such suffix code having four terminal zeroes is appended with a 1, and the initial part of the run-length code has a maximum of seven zeroes.

The blob parameter codes are specifically constructed to obey the rules above. In addition, they may be expected to result in efficiencies for Group 4 machines, where HDLC procedures are in effect. In this case the pattern of 01111110 is reserved for a synchronization code for data frames. To avoid occurrence of this pattern in the data stream requires insertion of 0 bits in runs of six or more 1's. In this respect we note that long runs of 1's in the blob parameter codewords are rare. Furthermore, combinations of the blob parameter codewords of five bits or less, which are the most frequently used codewords, will never result in six contiguous 1's.

In Group 3 machines, resynchronization is required upon detection of any of a number of error conditions. Among these are the following :

a) A HEAD code occurs and the corresponding run extends to the right of the image line of for example 1728 pels.

b) A CONNECT code occurs, and the corresponding run extends to the left of and/or to the right of the image line.

c) A HEAD or CONNECT code occurs, and the corresponding run overlaps, or, is not offset by at least one pel from the adjacent run, if any, on the left in the same image line.

d) A CONNECT code occurs and there is no candidate run in the previous line to which a connection can be made.

e) A CONNECT code occurs and the computed right end of the corresponding run precedes the computed left end of the run. This occurs, for example, if a run of three pels in one image line is to connect to a run of one pel in the next line with a CONNECT (LHS = 1, RHS = -1), but the connection is erroneously made to a run of one pel in the first image line.

f) An END code occurs, and there is no candidate run in the previous line to which the END code may apply.

g) An unassigned supplementary code is detected.

Upon detection of an error condition, the decoder searches for the next EOL code of eleven zeroes and a 1. The code between the detected error and the EOL code is bypassed, and any remaining portion of the image line under construction is filled with the corresponding portion of the previous scan line. This permits the possibility that the initial portions of the subsequent lines up to the next restart point are faithfully captured.

2.2.7 Error Handling

In Group 3 machines, we may restrict error propagation by restarting the coding algorithm every K lines. The K factor may be set equal to 2 or 4 for the standard normal and higher resolutions respectively. Of importance, the restart can be made adaptive according to line conditions. The encoder can change the K factor at will without having to transmit any indication of the K factor to the decoder. A restart simply causes the black runs on the next line to be all heads of blobs. This is a condition which may occur normally, and the decoder need not know that such condition is caused by a restart.

In Group 4 machines, it is normally assumed that an error free code transmission environment will be established by a level two protocol procedure such as HDLC. For such an environment it is most efficient to use a K factor of infinity and to use the wrap-around coding procedure. There are, however, rare situations where errors can escape level two procedures. Thus, the designers of the X.25 procedure chose to check for frames out of sequence at level three although this is usually completely solved at level two. An undetected frame out of sequence could ruin the integrity of the X.25 procedure. Additional error detection can optionally be built into the coded facsimile data by the simple addition of a black stripe one pel wide at the right hand side of the image. The stripe must be offset from the real image data by three white pels so that the stripe will code as one blob with (0,0) CONNECTS. This stripe uses up only a few pels of the image field which is normally unimportant. It can be added by an image preprocessor and is not linked to the actual coding algorithm.

In the worst case only, the black stripe will add 2N bits to the coded data, where N is the number of scan lines in the image. In actual practice the number of additional bits is considerably less. In our experiments we determined that the addition of the black stripe resulted in 5941 additional bits to code all eight CCITT test images, or, less than 750 additional bits per image on the average. This assumes digitization at full resolution and a K factor of infinity. Of interest, for image No. 1, addition of the black stripe resulted in 5476 less bits to code the image. The improvement results because of the containment of the blob position parameter to one line, and is particularly effective for sparse images. The stripe provides a simple consistency check at the decoder that no errors have slipped past level two procedures into the facsimile code stream. If an error is detected by the loss of the black stripe, the decoding process can be check pointed, and a recovery procedure initiated. The stripe, of course, can be shielded from actual printing. In addition, the stripe limits white

and black runs in the image plane to one scan line which may be an advantage for simple facsimile encoders and decoders. We feel the black stripe is a very useful adjunct to the basic code. However, we feel its addition should be left to the individual manufacturer. Our compression results do not reflect the use of the black stripe.

2.2.8 Extension of Line Length

For lines of any length greater than 2560 pels, provision is made for head lengths, L, or displacements, D, equal to or greater than 2560 as follows. Excess displacements may be coded as previously indicated for excess displacements resulting from the "wrap-around" permissible in an error free environment. Specifically, divide the given quantity by 2560, giving an integral quotient, Q, and a remainder, R, less than 2560. The codeword corresponding to the value of 2560 is issued Q times, followed by the codeword for the remainder, R. For displacements, D, the codewords used are those in the standard one-dimensional white-run-length Modified Huffman codebook. In this case, if R is zero, the codeword for run length zero is issued.

For excess head lengths we may follow a similar procedure. Here we divide the given head length by 2046, again giving an integral quotient, Q, and a remainder, R, less than 2046. In this case we first issue the Huffman code for the third category of "long" heads. Next, the 11 bit binary value of 1979 is issued. Next the 11 bit binary value of 1 is issued Q-1 times. Finally we issue the 11 bit binary value of the remainder, R, plus 2. Note that the value of 1979 is different from any other 11 bit long head suffix, which has the maximum value of 2560 minus 582, or 1978. The decoder may therefore uniquely interpret the value of 1979 to indicate an excess head length condition. Upon the first occurrence of this value, the decoder continues to inspect successive 11 bit segments, until one occurs that does not have the value 1. The total number of 11 bit segments with the value of 1 is Q-1. The first occurring 11 bit segment with value greater than 1 is the remainder, R, plus 2. The addition of 2 prevents the final 11 bits from assuming a value of all zeroes, which is reserved for the EOL code, or a value of 1, which is reserved for the quotient count.

3. CODING EFFICIENCY

In Tables 1 to 6, we show the results of a computer simulation of the Frank coding scheme for the eight CCITT Study Group XIV test documents of the 1976 Graphics Coding Contest. We display the number of bits, compression factors, and transmission times for each individual image; we also show the average results for the ensemble of the eight images. These eight images were horizontally scanned at a resolution of 8x8 pels/mm. Each image contains 1728 pels per scan line and 2128 scan lines per image. All the results are based on a transmission rate of 4800 bits per second. We use the blob parameter codebook, based on K=infinity, shown in Appendix B.

Tables 1 to 3 contain results for the individual images. The left-most three columns contain the figures for images coded at the full resolution of 8x8 pels/mm; the right-most three columns contain the figures for images coded at the resolution of 8x4 pels/mm, where every even numbered scan line has been omitted from the coding procedure. Under each resolution we show three columns labeled 0 ms, 5 ms, and 10 ms, indicating a minimum transmission time per image line of 0, 5, and 10 milliseconds, respectively. The numbers in the 0 ms columns include only the code bits and the return to control bits, and represent therefore the raw efficiencies for Group 3 machines. The figures under the 5 ms and 10 ms headings contain the code bits plus the bits for the start of message (one EOL), fill, EOL, and return to control. For each image we show the results when the K parameter equals 2, 4, and infinity.

In Table 1 we display the total number of bits to code each image. Table 2 contains the compression factors, which are defined as the number of pels per image divided by the number of bits to code the image. With a resolution of 8x8 pels/mm, the number of pels equals 1728x2128 = 3,677,184; with a resolution of 8x4, the number of pels equals 1728x1064 = 1,838,592. In Table 3 we show the transmission times, which are calculated by dividing the number of bits from Table 1 by 4800 bits per second.

Tables 4 to 6 contain the average figures of the individual image results reported in Tables 1 to 3. In Table 4 we show the average number of bits required to code each of the eight images. Table 5 contains the average compression factors, which are calculated by dividing the number of pels per image by the average number of bits, as shown in Table 4. In Table 6 we show the average transmission times.

4. PERFORMANCE WITH RESPECT TO SELECTION CRITERIA

AT&T presented a paper, "Selection Criteria for Advanced Facsimile Encoding Schemes," at the December 1978 meeting of CCITT Study Group (D16). There was also a United States contribution, "Criteria for the Evaluation of Two-Dimensional Coding Techniques for Use in Digital Facsimile Terminals," (D17). These two documents were primary inputs to the accepted standard method of evaluating two-dimensional coding schemes. The purpose of this section is to address in part the issues put forth in the selection criteria documents. Answers to remaining points will be submitted during the testing phase of the evaluation process, in line with the December 1978 agreement. We now address the points specifically raised in the AT&T selection criteria paper.

1. A new standard coding scheme should be designed where possible to be an extension of existing standard coding schemes.

   The Frank code is a direct upward extension of the standard one-dimensional Modified Huffman Code algorithm. The Frank algorithm collapses directly to a one-dimensional run-length algorithm if no CONNECT codes are allowed. The codebook used for the blob displacements is exactly the same as that used for the white runs in the Modified Huffman code. One advantage we feel this code has over other current two-dimensional proposals before CCITT Study Group XIV is that all additional codes are incorporated in one uniform codebook, and that this codebook has been optimized for the event frequencies of the blob parameters.

2. A new standard coding scheme should be easy and economical to implement in both dedicated hardware and in sequential machines such as minicomputers.

   The Frank code has been widely studied and used in the Bell System. It has been implemented in a variety of applications. One of these is the encoding of trademarks and graphics for photocomposing advertisements for the Yellow Pages telephone directories, and another is as a preprocessor for encoding engineering documents in vector form [1,2,3,4]. It has been implemented in hardwired logic [6], in microprocessors, in minicomputers, and in main frame computers. Implementation is direct and straightforward. There is a clear, clean price performance spectrum between the various implementations. We feel certain that manufacturers with different system design objectives will have no problem using this code in their terminals.

Use of the proposed coding scheme may require a patent license obtainable from the Western Electric Company, Greensboro, North Carolina, USA. Such licenses would be granted to all parties at a reasonable royalty.

3. A new standard coding scheme should be based on the most statistically relevant events.

After experimentation with a wide class of images, we have found that the Huffman coded blob parameters are a minimum efficient statistically relevant set. This can be seen in part from the frequency data presented with the final ensemble codebook. The limit of skew displacements to three in determining CONNECTS allows the algorithm to capture almost all image features which are of statistical importance. It also leads to a clear partitioning of all features into separate blobs. The resultant Huffman code is both short and manageable. Truncation of the skew displacement at three is more effective than allowing other values. Displacements as large as a full page width are of little statistical importance in the coding procedure, and displacements as little as 1 loose important relevant events. Appendix A includes further discussion of the advantages of a displacement of three.

4. A new standard coding scheme should exploit Huffman coding wherever possible, and the scheme should permit linguistic extensions of the Huffman code.

The Frank code algorithm fully utilizes Huffman coding for HEADS, CONNECTS, and ENDS. It is strongly recommended that 12 additional unassigned codewords be added which will permit future extensions of the coding scheme. Examples of the utility of this provision are to be found in the AT&T selection criteria paper.

5. A new standard coding scheme should be suitable for use in terminals which have positive error control, as well as in those which do not.

Almost all two-dimensional coding algorithms are most efficient if they are allowed to operate in an error free environment. However, for G3 class machines this is not the case and adjustments must be made to the algorithm to prevent errors from totally corrupting the encoded message. The Frank code accomplishes this by restarting the two-dimensional part of the algorithm periodically. This is covered earlier in this paper. Of particular interest, this algorithm restart can be made adaptive. In fact, the encoder can restart the al-

gorithm at will without having to notify the decoder of the K factor.

6. A new standard coding scheme should lend itself to the addition of operations for feature extraction to permit the design of future upward compatible coding schemes based on higher level image structures, including character recognition.

   The Frank code is based on image features. The HEAD, CONNECT, and END description of an image is very similar to a chain-vector description of an image border. A sizable body of literature exists which shows how such an image description can be used to classify higher level image features, ranging from stroke data, to simple constructs, and up to full characters. The specific use of this code to obtain vector representations of engineering documents was discussed previously [2].

7. A new standard coding scheme should allow overlap processing and multiple pass processing of the data to effect coding and decoding.

   The translation of a pel-by-pel description of an image into a HEAD, CONNECT, and END description is totally independent of the process which assigns codewords to these language primitives. Similarly, the separation of the coded bit stream into codewords and the decoding of the codewords is independent of the image reconstruction process. This clearly meets the overlap processing objective.

8. A new standard coding scheme should permit natural growth from handling bilevel material to future extensions for handling multilevel gray scale, color, and texture parameters.

   The Frank code algorithm can easily be extended to code a structure like a contour map, where there is "step" information as well as the image contour information. A description of how to code images consisting of areas of varying content, such as gray scale, color, or texture, has been published [5]. This is a direct upward extension from coding bilevel images with Frank coding.

(2595)

REFERENCES

1] Frank, A. J., "High fidelity encoding of two-level, high resolution images," Conference Record, IEEE International Conference on Communications, June 1973, Session 26, pp. 5-10.

2] Pferd, W., and Ramachandran, K., "Computer aided automatic digitizing of engineering drawings," Proceedings, IEEE Second International Computer Software and Applications Conference, November 1978, pp. 630-635.

3] Denes, P. B., and Gershkoff, I. K., "An interactive system for page layout design," Proceedings of the ACM Annual Conference, November 1974, pp. 212-221.

4] Frank, A. J., and Groff, E. H., "On statistical coding of two-tone image ensembles," Proceedings of the Society for Information Display, Volume 17, Number 2, Second Quarter, 1976, pp. 102-110.

5] Frank, A. J., "Partitioning and coding a two-dimensional field," Proceedings, Third International Conference on Pattern Recognition, November 1976, pp. 816-821.

6] Todd, R. G., "A hardware decoder for two-dimensionally compressed pictures," Master's Thesis, Massachusetts Institute of Technology, 1975.

Appendices : 3

APPENDIX A

OPTIMAL LINE-TO-LINE RUN DISPLACEMENT

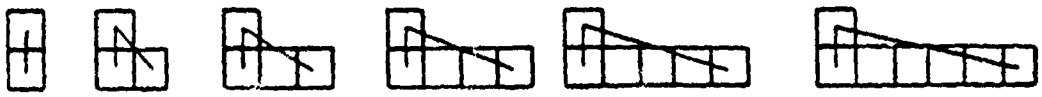
Frank coding specifies the maximum of the line-to-line run displacement to be three. This condition applies separately to both the left ends and the right ends of two runs on successive lines of a blob. It is this containment on both sides which establishes the blob structures and leads to the advantages afforded by a feature-oriented approach. With a displacement of three, there are seven ways in which the left ends of two successive runs may connect, and similarly there are seven way in which the right ends connect. We have shown that in typical facsimile material, the displacements on one side of a blob are correlated with the displacements on the other side. To capture this dependency blob coding uses a single CONNECT code to specify the connection on both sides. There are 7X7 or 49 possible values to the CONNECT code.

We may ask whether the choice of a displacement of three is optimal for the Frank code. If this is decreased to two, the number of distinct CONNECT codes drops to 5X5 or 25, giving smaller Huffman codewords. However, this improvement is offset by a larger number of blobs. Similarly, an increase in the displacement to 4 requires 9X9 or 81 distinct CONNECT codes, giving larger codewords, but results in fewer blobs. Other choices are also possible.

We show the displacements in the range 0-5 in the figure below. Under each construct we list the absolute value of the maximum displacement, D, and the number of signed displacements, 2D+1. The resulting contour at the right side of a blob is represented by a straight line connecting the midpoints of the end pels. This line forms an angle, A, with the vertical, indicating the maximum contour slopes for a given D. With D equal to 1, A is 45 degrees. Contours exceeding this angle result in multiple blobs, each containing only one run. This multiplicity can be reduced by increasing D. Listed in the figure are the values of A as well as the first differences in A. The rate of increase in A diminishes as D increases. Progressing from a D of 0 to a D of 1 improves A by 45 degrees. In going from a D of 1 to a D of 2 this improvement drops to 18.4 degrees. We have noted that contour lines tend to be more vertical than horizontal, resulting in heavier weighting of smaller D values. In our studies, we determined a displacement of three or two gives optimal coding efficiency. On the CCITT test images the difference in efficiency resulting from these two choices of the displacement is negligible. Thus, the ensemble of the eight CCITT test images with full resolution and a K factor of infinity requires a total of

1,976,856 bits using a displacement of two, and a total of 1,973,945 bits using a displacement of three. Although the displacement of three resulted in a total smaller number of bits, the displacement of two excelled in half of the images. However, a displacement of three results in fewer blobs than a displacement of two. With the expectation that pattern recognition tasks may be facilitated by the smaller number of blobs we have chosen a displacement of three.

Thus, a displacement of three insures that both coding efficiency is attained, and that blobs are defined which are of meaningful size and coherence for recognition of stroke or other micropatterns, as may be desired. We believe that the strong advantages afforded by the pattern recognition capabilities of the Frank code warrant its adoption even if slight inefficiencies result.



D	0	1	2	3	4	5
2D+1	1	3	5	7	9	11
λ	0.0	45.0	63.4	71.6	76.0	78.7
λ(n+1)−λ(n)		45.0	18.4	8.2	4.4	2.7

APPENDIX B

Blob parameter codebook

Parameter connects	Frequency*)	Codeword Length	Binary Codeword
-3,-3	220	11	11000001001
-3,-2	248	10	1100010111
-3,-1	333	10	1100010110
-3, 0	721	9	110010111
-3,+1	328	10	1100010101
-3,+2	382	10	1100010100
-3,+3	303	10	1100010011
-2,-3	285	10	1100010010
-2,-2	821	9	110010110
-2,-1	1444	8	11010111
-2, 0	1767	7	1110111
-2,+1	1461	8	11010110
-2,+2	1443	8	11010101
-2,+3	472	9	110010101
-1,-3	504	9	110010100
-1,-2	1957	7	1110110
-1,-1	11884	5	00011
-1, 0	16623	4	1011
-1,+1	6186	6	111111
-1,+2	1544	8	11010100
-1,+3	416	10	1100010001
0,-3	705	9	110010011
0,-2	1684	8	11010011
0,-1	17351	4	1010
0, 0	104746	2	01
0,+1	16066	4	1001
0,+2	1876	7	1110101
0,+3	918	8	11010010
+1,-3	372	10	1100010000
+1,-2	1410	8	11010001
+1,-1	5351	6	111110
+1, 0	15923	4	1000
+1,+1	8984	6	000001
+1,+2	1916	7	1110100
+1,+3	694	9	110010010
+2,-3	409	10	1100001111
+2,-2	1108	8	11010000
+2,-1	1222	8	11001111
+2, 0	1965	7	1110000
+2,+1	1611	8	11001110
+2,+2	717	9	110010001
+2,+3	282	10	1100001110
+3,-3	306	10	1100001101
+3,-2	306	10	1100001100
+3,-1	366	10	1100001011
+3, 0	904	8	11001101

(----)

APPENDIX B
(Cont.)

Parameter	Frequency*)	Codeword Length	Binary Codeword
+3,+1	559	9	110010000
+3,+2	252	10	1100001010
+3,+3	224	11	11000001000
ends	20734	4	0011
head lengths			
1	6879	6	111101
2	10896	5	00001
3	15524	4	0010
4	9706	5	00010
5	4860	6	111100
6	3480	7	1110010
7	2864	7	1110001
8	2687	7	1110011
9	2990	7	1101111
10	2820	7	1101110
11	2390	7	1101101
12	2143	7	1101100
13	1399	8	11001100
14	789	9	110001111
15	669	9	110001110
16	489	9	110001100
17	412	10	1100001001
18	359	10	1100001000
19	283	10	1100000111
20	251	10	1100000110
21	301	10	1100000101
22	208	11	11000000111
23	183	11	11000000110
24	146	11	11000000100
25	91	12	110000000111
26	80	12	110000000110
27	61	13	1100000001001
28	43	13	1100000001000
29	44	13	1100000000111
30	31	14	11000000000110
31	32	13	1100000000100
32	32	14	11000000000111
33	40	13	1100000000110
34	25	15	110000000000111
35	38	13	1100000000101
36	29	14	11000000000101
37	18	15	110000000000110
38	12	15	110000000000100
39	22	14	11000000000100
40	14	15	110000000000101
41-71	207	11(+ 5 suffix)	11000000101
72-582	753	9(+ 9 suffix)	110001101
583-2560	116	12(+11 suffix)	110000000101

*) Note : Total of the eight images at a resolution of 8x8 pels/mm and k=infinit

APPENDIX C

BICB PARAMETER CODEBOOK WITH SUPPLEMENTARY CODES

Parameter connects	Frequency*)	Codeword Length	Binary Codeword
-3,-3	220	11	11000001001
-3,-2	248	10	1100010111
-3,-1	333	10	1100010110
-3, 0	721	9	110010111
-3,+1	328	10	1100010101
-3,+2	382	10	1100010100
-3,+3	303	10	1100010011
-2,-3	285	10	1100010010
-2,-2	821	9	110010110
-2,-1	1444	8	11010111
-2, 0	1767	7	1110111
-2,+1	1461	8	11010110
-2,+2	1443	8	11010101
-2,+3	472	9	110010101
-1,-3	504	9	110010100
-1,-2	1957	7	1110110
-1,-1	11884	5	00011
-1, 0	16623	4	1011
-1,+1	6186	6	111111
-1,+2	1544	8	11010100
-1,+3	416	10	1100010001
0,-3	705	9	110010011
0,-2	1684	8	11010011
0,-1	17351	4	1010
0, 0	104746	2	01
0,+1	16066	4	1001
0,+2	1876	7	1110101
0,+3	918	8	11010010
+1,-3	372	10	1100010000
+1,-2	1410	8	11010001
+1,-1	5351	6	111110
+1, 0	15923	4	1000
+1,+1	8984	6	000001
+1,+2	1916	7	1110100
+1,+3	694	9	110010010
+2,-3	409	10	1100001111
+2,-2	1108	8	11010000
+2,-1	1222	8	11001111
+2, 0	1965	7	1110000
+2,+1	1611	8	11001110
+2,+2	717	9	110010001
+2,+3	282	10	1100001110
+3,-3	306	10	1100001101
+3,-2	306	10	1100001100
+3,-1	366	10	1100001011
+3, 0	904	8	11001101

APPENDIX C

(Cont.)

Parameter	Frequency*)	Codeword Length	Binary Codeword
+3,+1	559	9	110010000
+3,+2	252	10	1100001010
+3,+3	224	11	11000001000
ends	20734	4	0011
head lengths			
1	6879	6	111101
2	10896	5	00001
3	15524	4	0010
4	9706	5	00010
5	4860	6	111100
6	3480	7	1110010
7	2864	7	1110001
8	2687	7	1110011
9	2990	7	1101111
10	2820	7	1101110
11	2390	7	1101101
12	2143	7	1101100
13	1399	8	11001100
14	789	9	110001111
15	669	9	110001110
16	489	9	110001100
17	412	10	1100001001
18	359	10	1100001000
19	283	10	1100000111
20	251	10	1100000110
21	301	10	1100000101
22	208	11	11000000111
23	183	11	11000000110
24	146	11	11000000100
25	91	12	110000000111
26	80	12	110000000110
27	61	13	1100000001001
28	43	13	1100000001000
29	44	13	1100000000111
30	31	15	110000000001111
31	32	14	11000000001001
32	32	14	11000000001000
33	40	13	1100000000110
34	25	15	110000000001100
35	38	13	1100000000101
36	29	15	110000000001110
37	18	16	1100000000010111
38	12	16	1100000000010101
39	22	15	110000000001101
40	14	16	1100000000010110
41-71	207	11(+ 5 suffix)	11000000101
72-582	753	9(+ 9 suffix)	110001101
583-2560	116	12(+11 suffix)	110000000101

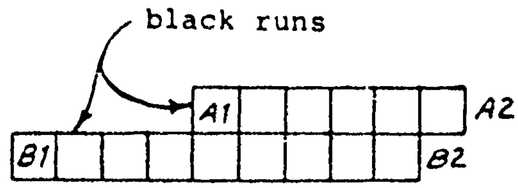
APPENDIX C

(Cont.)

Parameter supplementary codes	Frequency*)	Codeword Length	Binary Codeword
1		16	1100000000010100
2		16	1100000000010011
3		16	1100000000010010
4		16	1100000000010001
5		16	1100000000001111
6		16	1100000000001110
7		16	1100000000001101
8		16	1100000000001100
9		16	1100000000001011
10		16	1100000000001010
11		16	1100000000001001
12		16	1100000000001000

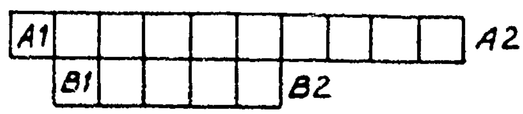
*) Note : Total of the eight images at a resolution of 8x8 pels/mm and k=infinity.

RULE	CODE ISSUED	POINTERS ADVANCED
1	HEAD	B



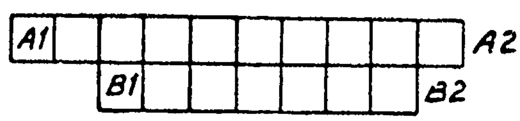
$A1 - B1 = 4 > 3$

RULE	CODE ISSUED	POINTERS ADVANCED
2	HEAD	B



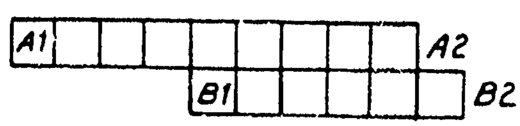
$|A1 - B1| = 1 \leq 3$, $A2 - B2 = 4 > 3$

RULE	CODE ISSUED	POINTERS ADVANCED
3	CONNECT	A and B



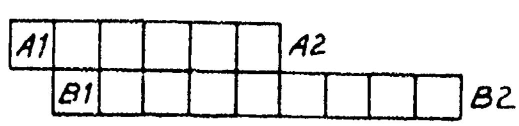
$|A1 - B1| = 2 \leq 3$, $|A2 - B2| = 1 \leq 3$

RULE	CODE ISSUED	POINTERS ADVANCED
4	END	A



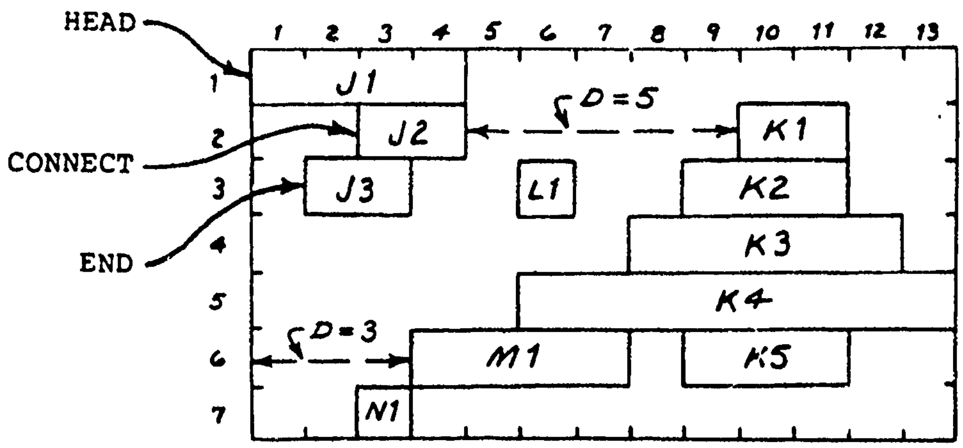
$B1 - A1 = 4 > 3$

RULE	CODE ISSUED	POINTERS ADVANCED
5	END	A



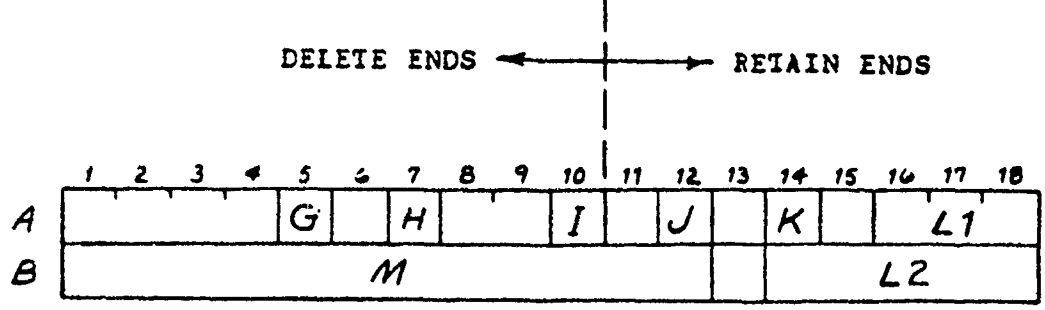
$|A1 - B1| = 1 \leq 3$, $B2 - A2 = 4 > 3$

Figure 1 – Examples of the five code rules



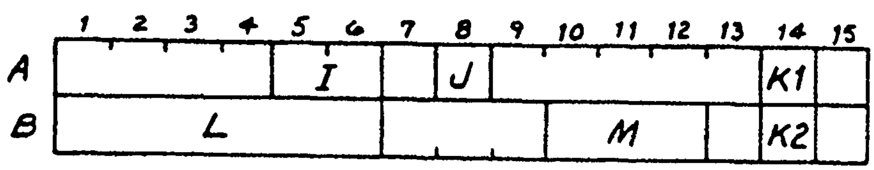
Line B	Runs compared Line A, Line B	Rule	Code Issued in Order Shown	Blob
1	null, J1	1	HEAD (L=4,D=0)	J
2	J1, J2	3	CONNECT (LHS=2,RHS=0)	J
	null, K1	1	HEAD (L=2,D=5)	K
3	J2, J3	3	CONNECT (LHS=-1,RHS=-1)	J
	K1, L1	1	HEAD (L=1,D=2)	L
	K1, K2	3	CONNECT (LHS=-1,RHS=0)	K
4	J3, K3	4	END	J
	L1, K3	5	END	L
	K2, K3	3	CONNECT (LHS=-1,RHS=1)	K
5	K3, K4	3	CONNECT (LHS=-2,RHS=1)	K
6	K4, M1	2	HEAD (L=4,D=3)	M
	K4, K5	3	CONNECT (LHS=3,RHS=-2)	K
7	M1, N1	2	HEAD (L=1,D=2)	N

Figure 2 - Example of code



Runs Compared Line A, Line B	Rule	Code Issued in Order Shown		Blob
G, M	1	HEAD	(L=12,D=0)	M
G, L2	4	END	redundant	G
H, L2	4	END	redundant	H
I, L2	4	END	redundant	I
J, L2	5	END		J
K, L2	5	END		K
L1, L2	3	CONNECT	(LHS=-2,RHS=0)	L

Figure 3 - The HEC end code redundancy



Runs compared Line A, Line B	Rule	Code Issued In Order Shown		Blob
I, L	1	HEAD	(L=6,D=0)	L
I, M	4	END	redundant	I
J, M	5	END	redundant	J
K1, M	1	HEAD	(L=3,D=3)	M
K1, K2	3	CONNECT	(LHS=0,RHS=0)	K

Figure 4 - The HEH end code redundancy

TABLE 1

Total bits

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
Image #1						
K=2	181530	215092	249513	99526	116349	133066
K=4	157485	191032	225998	91724	108413	125194
K=infinity	135229	166233	201721	85715	101099	117967
Image #2						
K=2	142434	171219	191544	78776	92949	101936
K=4	111275	140453	163736	67169	81474	91246
K=infinity	80958	110009	136023	55775	69949	80520
Image #3						
K=2	326643	354801	370338	177034	190870	198426
K=4	264849	293123	311310	153906	167719	176480
K=infinity	203581	231368	252309	130629	144119	154092
Image #4						
K=2	643252	671953	687340	353632	367808	375139
K=4	566162	594515	610054	330009	343976	351238
K=infinity	489000	516358	532182	306975	320345	327593
Image #5						
K=2	345341	373627	389649	187601	201619	209034
K=4	285892	314287	332893	165407	179393	187821
K=infinity	227191	255328	276414	143652	157437	166843
Image #6						
K=2	241392	270143	284122	129166	143430	150112
K=4	181594	210473	225061	103455	117679	124447
K=infinity	122617	150766	165889	78014	91837	98690
Image #7						
K=2	712171	739961	748153	386206	399843	403943
K=4	626546	654635	662932	358465	372104	376268
K=infinity	541518	569250	577693	330806	344090	348314
Image #8						
K=2	325243	354418	359130	175415	189946	191999
K=4	249604	280180	285640	144560	159702	161967
K=infinity	173851	205589	211717	113450	129143	131720

TABLE 2

Compression factors
(pels/bits)

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
Image #1						
K=2	20.257	17.096	14.737	18.473	15.802	13.815
K=4	23.349	19.249	16.271	20.045	16.959	14.686
K=infinity	27.192	22.121	18.229	21.450	18.186	15.58[illegible]
Image #2						
K=2	25.817	21.477	19.198	23.339	19.781	18.037
K=4	33.046	26.181	22.458	27.373	22.567	20.150
K=infinity	45.421	33.426	27.034	32.964	26.285	22.634
Image #3						
K=2	11.258	10.364	9.929	10.386	9.633	9.266
K=4	13.884	12.545	11.812	11.946	10.962	10.418
K=infinity	18.063	15.893	14.574	14.075	12.757	11.932
Image #4						
K=2	5.717	5.472	5.350	5.199	4.999	4.901
K=4	6.495	6.185	6.028	5.571	5.345	5.235
K=infinity	7.520	7.121	6.910	5.989	5.739	5.612
Image #5						
K=2	10.648	9.842	9.437	9.801	9.119	8.796
K=4	12.862	11.700	11.046	11.116	10.249	9.789
K=infinity	16.185	14.402	13.303	12.799	11.678	11.020
Image #6						
K=2	15.233	13.612	12.942	14.234	12.819	12.248
K=4	20.249	17.471	16.339	17.772	15.624	14.774
K=infinity	29.989	24.390	22.167	23.567	20.020	18.630
Image #7						
K=2	5.163	4.969	4.915	4.761	4.598	4.552
K=4	5.869	5.617	5.547	5.129	4.941	4.886
K=infinity	6.791	6.460	6.365	5.558	5.343	5.279
Image #8						
K=2	11.306	10.375	10.239	10.481	9.680	9.576
K=4	14.732	13.124	12.873	12.719	11.513	11.352
K=infinity	21.151	17.886	17.368	16.206	14.237	13.958

(2595)

TABLE 3

Transmission times
(seconds/page)*)

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
Image #1						
K=2	37.82	44.81	51.98	20.73	24.24	27.73
K=4	32.81	39.80	47.08	19.11	22.59	26.08
K=infinity	28.17	34.63	42.03	17.86	21.06	24.58
Image #2						
K=2	29.67	35.67	39.91	16.41	19.36	21.24
K=4	23.18	29.26	34.11	13.99	16.97	19.01
K=infinity	16.87	22.92	28.34	11.62	14.57	16.78
Image #3						
K=2	68.05	73.92	77.15	36.88	39.76	41.34
K=4	55.18	61.07	64.86	32.06	34.94	36.77
K=infinity	42.41	48.20	52.56	27.21	30.02	32.10
Image #4						
K=2	134.01	139.99	143.20	73.67	76.63	78.15
K=4	117.95	123.86	127.09	68.75	71.66	73.17
K=infinity	101.88	107.57	110.87	63.95	66.74	68.25
Image #5						
K=2	71.95	77.84	81.18	39.08	42.00	43.55
K=4	59.56	65.48	69.35	34.46	37.37	39.13
K=infinity	47.33	53.19	57.59	29.93	32.80	34.76
Image #6						
K=2	50.29	56.28	59.19	26.91	29.88	31.27
K=4	37.83	43.85	46.89	21.55	24.52	25.93
K=infinity	25.55	31.41	34.56	16.25	19.13	20.56
Image #7						
K=2	148.37	154.16	155.87	80.46	83.30	84.15
K=4	130.53	136.38	138.11	74.68	77.52	78.39
K=infinity	112.82	118.59	120.35	68.92	71.69	72.57
Image #8						
K=2	67.76	73.84	74.82	36.54	39.57	40.00
K=4	52.00	58.37	59.51	30.12	33.27	33.74
K=infinity	36.22	42.83	44.11	23.64	26.90	27.44

*) Note : Based on a transmission rate of 4800 bps.

TABLE 4

Average number of bits per image

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
All Images						
K=2	364751	393902	409974	198420	212852	220459
K=4	305426	334837	352203	176837	191308	199333
K=infinity	246743	275613	294244	155627	169752	178217

TABLE 5

Average compression factors

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
All Images						
K=2	10.081	9.335	8.969	9.266	8.638	8.340
K=4	12.040	10.982	10.441	10.397	9.611	9.224
K=infinity	14.903	13.342	12.497	11.814	10.831	10.317

TABLE 6

Average transmission times
(seconds/page)*)

	Full Resolution of 8x8			Half Resolution of 8x4		
	0 ms	5 ms	10 ms	0 ms	5 ms	10 ms
All Images						
K=2	75.99	82.06	85.41	41.34	44.34	45.93
K=4	63.63	69.76	73.38	36.84	39.86	41.53
K=infinity	51.40	57.42	61.30	32.42	35.37	37.13

*) Note : Based on a transmission rate of 4800 bps.

Note : The figures in the 0 ms columns include only code bits and return to control bi These results represent the raw efficiencies for Group 3 machines. The figures in the 5 ms and 10 ms columns contain code bits plus bits for start of message, fill, EOL, and return to control.

(2595)

TITLE: Correction and Addition to the AT&T Proposal for Two-Dimensional Facsimile Coding Scheme, submitted March 28, 1979.

1. Change the binary codeword for the head length of 39 in Appendix C as follows:

   Was: 110100000001101

   Should be: 110000000001101

2. Add the following to section 2.2.6, Resynchronization.

   In Group 3 machines, resynchronization is required upon detection of any of a number of error conditions. Among these are the following.

   (a) A HEAD code occurs and the corresponding run extends to the right of the image line of for example 1728 pels.

   (b) A CONNECT code occurs, and the corresponding run extends to the left of and/or to the right of the image line.

   (c) A HEAD or CONNECT code occurs, and the corresponding run overlaps, or, is not offset by at least one pel from the adjacent run, if any, on the left in the same image line.

   (d) A CONNECT code occurs and there is no candidate run in the previous line to which a connection can be made.

   (e) A CONNECT code occurs and the computed right end of the corresponding run precedes the computed left end of the run. This occurs, for example, if a run of three pels in one image line is to connect to a run of one pel in the next line with a CONNECT (LHS = 1, RHS = -1), but the connection is erroneously made to a run of one pel in the first image line.

   (f) An END code occurs, and there is no candidate run in the previous line to which the END code may apply.

(g) An unassigned supplementary code is detected.

Upon detection of an error condition, the decoder searches for the next EOL code of eleven zeroes and a 1. The code between the detected error and the EOL code is bypassed, and any remaining portion of the image line under construction is filled with the corresponding portion of the previous scan line. This permits the possibility that the initial portions of the subsequent lines up to the next restart point are faithfully captured.

APPENDIX F

SUBROUTINES WHICH ARE COMMON TO ALL ALGORITHMS

```
START OF DCEC UPRINT PROGRAM          DSNAME=D0031.REDTAP.FORT
C      PROGRAM REDTAP32
C
       IMPLICIT INTEGER(A-Z)
       INTEGER PELBUF(1500),OTBUF(60)
       DATA PELMAX,PELFIL,OTFIL,TERM/1728,1,2,5/
C
C*************************** BEGIN PROGRAM ****************************
C
       INLNCT=0
  150  CONTINUE
       DO 100 I=1,60
  100  OTBUF(I)=0
       ID=1
       IF=250
       READ(PELFIL,300,END=500) IC,J
  300  FORMAT(250I4)
       J1=J
  316  IF(J.GT.250) GO TO 315
       JID1=J+ID-1
       READ(PELFIL,300) (PELBUF(K),K=ID,JID1)
       GO TO 400
  315  CONTINUE
       READ(PELFIL,300) (PELBUF(K),K=ID,IF)
       ID=IF+1
       IF=IF+250
       J=J-250
       IF(J.EQ.0) GO TO 400
       GO TO 316
  400  CONTINUE
       IF(INLNCT.GT.200) GO TO 450
C      WRITE(TERM,410) IC,J1
  410  FORMAT(5X,I4,5X,I6)
C      WRITE(TERM,420) (PELBUF(K),K=1,J1)
  420  FORMAT(2X,20(I4,2X))
  450  CONTINUE
       OTELP=1
       DO 460 I=1,J1
       RUN=PELBUF(I)
       IF(RUN.EQ.0) GO TO 700
       DO 470 K=1,RUN
       CALL MI2B(IC,OTBUF,OTELP,1)
       OTELP=OTELP+1
       IF(OTELP.GT.PELMAX) GO TO 480
  470  CONTINUE
       IC=MOD(IC+1,2)
  460  CONTINUE
  480  CONTINUE
       INLNCT=INLNCT+1
       WRITE(OTFIL) INLNCT,PELMAX,OTBUF
       GO TO 150
  500  CONTINUE
       WRITE(TERM,510) INLNCT,INLNCT
  510  FORMAT('0LINES WRITTEN =',I6,'; LAST LINE NUMBER =',I6)
       STOP
  600  CONTINUE
       STOP 600
  700  CONTINUE
       STOP 700
       E N D
0           END OF DCEC UPRINT PROGRAM          LINES PRINTED=     59
```

```
START OF DCEC JPRINT PROGRAM          DSNAME=D0031.CCDELN.FORT
      SUBROUTINE CODELN(LENGTH,POLAR,CDELCT,CDDATA)
C
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C
C******************************* BEGIN PROGRAM *********************
C
C    INITIALIZE MAKE UP CODE, MAKE UP CODE LENGTH
C
      MCODE=0
      MLENG=0
C
C    CHECK INPUTS
C
      IF(POLAR.LT.1.OR.POLAR.GT.2) CALL EXIT
      IF(LENGTH.LT.0.OR.LENGTH.GT.1728) CALL EXIT
C
      IF(LENGTH.LE.63) GO TO 10
C
C    CALCULATE MAKE UP CODE INDEX, CODE, LENGTH
C     AND WRITE TO CODE LINE
C
      INDEX=LENGTH/64+64
      MCODE=CODE(3,INDEX,POLAR)
      MLENG=CODE(1,INDEX,POLAR)
      CALL MI2B(MCODE,CDBUF,CDELCT+1,MLENG)
      CDELCT=CDELCT+MLENG
      CDDATA=CDDATA+MLENG
C
C    CALCULATE TERMINATING CODE INDEX, CODE, LENGTH
C     AND ADD TO CODE LINE
C
   10 CONTINUE
      INDEX=MOD(LENGTH,64)+1
      TCODE=CODE(3,INDEX,POLAR)
      TLENG=CODE(1,INDEX,POLAR)
      CALL MI2B(TCODE,CDBUF,CDELCT+1,TLENG)
      CDELCT=CDELCT+TLENG
      CDDATA=CDDATA+TLENG
C
      RETURN
      END
```

```
      SUBROUTINE STATS(LENGTH,INLNCT,DIAG)
      IMPLICIT INTEGER(A-Z)
C
      INTEGER MTT(5),ITT(2,5),LENGTH(INLNCT)
      REAL STT(2,5),SUM,SUMSQ
      LOGICAL DIAG
C********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,EFFIL
C
      DATA MTT/0,24,48,96,192/
C
C************************BEGIN PROGRAM*************************************
C
      DO 300 I=1,5
      ITT(1,I)=10000
      ITT(2,I)=0
      SUM=0.
      SUMSQ=0.
      DO 100 J=1,INLNCT
C
C   FIND FILLED LINE LENGTH
C
      LEN=MAX0(LENGTH(J),MTT(I))
      IF(DIAG) WRITE(TERM,50) LEN
   50 FORMAT(I8)
C
C   FIND MINIMUM LINE LENGTH
C
```

UNCLASSIFIED

UNCLASSIFIED

```
      ITT(1,I)=MIN0(LEN,ITT(1,I))
C
C   FIND MAXIMUM LINE LENGTH
C
      ITT(2,I)=MAX0(LEN,ITT(2,I))
C
C   FIND SUM OF LENGTHS
C
      SUM=SUM+FLOAT(LEN)
      SUMSQ=SUMSQ+(FLOAT(LEN))**2
  100 CONTINUE
C
C   FIND SAMPLE MEAN AND STANDARD DEVIATION
C
      STT(1,I)=SUM/FLOAT(INLNCT)
      STT(2,I)=SQRT((SUMSQ-(SUM**2)/FLOAT(INLNCT))/FLOAT(INLNCT-1))
  300 CONTINUE
C
      WRITE(LPFIL,400)(ITT(1,I),I=1,5)
  400 FORMAT(
     *'0                         MINIMUM TRANSMISSION TIME (4800 BPS)'//
     *'   CODED LINE'/
     *'     LENGTH               0 MS     5 MS    10 MS    20 MS    40 MS'/
     *'   STATISTICS:'//
     *'     MINIMUM',10X,5(I8)//)
      WRITE(LPFIL,410)(ITT(2,I),I=1,5)
  410 FORMAT(
     *'     MAXIMUM',10X,5(I8)//)
      WRITE(LPFIL,420)(STT(1,I),I=1,5)
  420 FORMAT(
     *'     SAMPLE MEAN',9X,5(F8.2)//)
      WRITE(LPFIL,430)(STT(2,I),I=1,5)
  430 FORMAT(
     *'     STANDARD DEVIATION',2X,5(F8.2))
C
      RETURN
      END
 0            END OF DCEC UPPRINT PROGRAM          LINES PRINTED=    200
```

```
      BLOCK DATA
C
      IMPLICIT INTEGER (A-Z)
C
C
      COMMON /G32BIT/KIBIT(32),KZBIT(32),LIBIT(32),LZBIT(32)
C
C
      DATA KIBIT /
     &               Z80000000,Z40000000,Z20000000,Z10000000,
     &               Z08000000,Z04000000,Z02000000,Z01000000,
     &               Z00800000,Z00400000,Z00200000,Z00100000,
     &               Z00080000,Z00040000,Z00020000,Z00010000,
     &               Z00008000,Z00004000,Z00002000,Z00001000,
     &               Z00000800,Z00000400,Z00000200,Z00000100,
     &               Z00000080,Z00000040,Z00000020,Z00000010,
     &               Z00000008,Z00000004,Z00000002,Z00000001/
C
      DATA KZBIT /
     &               Z7FFFFFFF,ZBFFFFFFF,ZDFFFFFFF,ZEFFFFFFF,
     &               ZF7FFFFFF,ZFBFFFFFF,ZFDFFFFFF,ZFEFFFFFF,
     &               ZFF7FFFFF,ZFFBFFFFF,ZFFDFFFFF,ZFFEFFFFF,
     &               ZFFF7FFFF,ZFFFBFFFF,ZFFFDFFFF,ZFFFEFFFF,
     &               ZFFFF7FFF,ZFFFFBFFF,ZFFFFDFFF,ZFFFFEFFF,
     &               ZFFFFF7FF,ZFFFFFBFF,ZFFFFFDFF,ZFFFFFEFF,
     &               ZFFFFFF7F,ZFFFFFFBF,ZFFFFFFDF,ZFFFFFFEF,
     &               ZFFFFFFF7,ZFFFFFFFB,ZFFFFFFFD,ZFFFFFFFE/
C
      DATA LIBIT /
     &               Z80000000,ZC0000000,ZE0000000,ZF0000000,
     &               ZF8000000,ZFC000000,ZFE000000,ZFF000000,
     &               ZFF800000,ZFFC00000,ZFFE00000,ZFFF00000,
     &               ZFFF80000,ZFFFC0000,ZFFFE0000,ZFFFF0000,
     &               ZFFFF8000,ZFFFFC000,ZFFFFE000,ZFFFFF000,
     &               ZFFFFF800,ZFFFFFC00,ZFFFFFE00,ZFFFFFF00,
     &               ZFFFFFF80,ZFFFFFFC0,ZFFFFFFE0,ZFFFFFFF0,
     &               ZFFFFFFF8,ZFFFFFFFC,ZFFFFFFFE,ZFFFFFFFF/
C
      DATA LZBIT /
     &               Z7FFFFFFF,Z3FFFFFFF,Z1FFFFFFF,Z0FFFFFFF,
     &               Z07FFFFFF,Z03FFFFFF,Z01FFFFFF,Z00FFFFFF,
     &               Z007FFFFF,Z003FFFFF,Z001FFFFF,Z000FFFFF,
     &               Z0007FFFF,Z0003FFFF,Z0001FFFF,Z0000FFFF,
     &               Z00007FFF,Z00003FFF,Z00001FFF,Z00000FFF,
     &               Z000007FF,Z000003FF,Z000001FF,Z000000FF,
     &               Z0000007F,Z0000003F,Z0000001F,Z0000000F,
     &               Z00000007,Z00000003,Z00000001,Z00000000/
C
C
C
C
      END
```

UNCLASSIFIED

```
START OF DCEC UPRINT PROGRAM              DSNAME=D0031.MI2B.FCRT
CMI2B
C
      SUBROUTINE MI2B(IVAL,IBA,JB,NB)
      IMPLICIT INTEGER(A-Z)
      DIMENSION IBA(2)
C
C******* MI2B MOVES THE BIT STRING RIGHT-JUSTIFIED IN IVAL
C        TO THE JB-TH THRU THE (JB+NB-1)-TH BIT OF IBA.
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
C
C******* MI2B EXECUTE *******
      JRHB=JB+NB-2
      NBT=NB
      JRE=JRHB/32+1
      JRB=MOD(JRHB,32)+1
      NBR=MIN0(NBT,JRB)
      LVAL=IVAL
      JIM=32-NBR
C
      J=LAND(LVAL,LZBIT(JIM))
      K=32-JRB
      LRE=LOR(LAND(IBA(JRE),LZBIT(JRB)),SHFTL(J,K))
      K=32-JIM
      LVAL=SHFTR(LVAL,K)
      NBT=NBT-JRB
C
  199 IF(NBT) 300,390,200
C
  200 IBA(JRE)=LRE
      JRE=JRE-1
      LRE=LVAL
      LVAL=0
      NBT=NBT-32
      GO TO 199
C
  300 JIM=-NBT
      LRE=LOR(LRE,LAND(IBA(JRE),LIBIT(JIM)))
  390 IBA(JRE)=LRE
      RETURN
C
      END
```

```
START OF UCEC UPRINT PROGRAM          DSNAME=D0031.I48.FCRT
CI48
      INTEGER FUNCTION I48(IBA,JB,NB)
      IMPLICIT INTEGER (A-Z)
      DIMENSION IBA(2)
C
C******* I48 RETURNS AN INTEGER VALUE FOR THE BIT STRING
C              STARTING AT THE JB-TH BIT OF IBA
C                 AND CONSISTING OF NB BITS.
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
C
C******* I48 EXECUTE ********
C
      IF(NB-1) 10,30,20
   10 STOP 10
   20 CONTINUE
      JRHB=JB+NB-2
      NBT=MINO(NB,32)
      JRE=JRHB/32+1
      JRB=MOD(JRHB,32)+1
      NBR=MINO(NBT,JRB)
      JIM=32-NBR
C
C    SHIFT RIGHT 32-JRB BITS AND PUT IN ZEROS ON LEFT
C
      J=IBA(JRE)
      K=32-JRB
      I48=LAND(LZBIT(JIM),SHFTR(J,K))
C
C    CALCULATE NUMBER OF BITS REMAINING IN LEFT PORTION IF ANY
      NBR=NBT-NBR
      IF(NBR.LE.0) RETURN
C
C    IF LEFT PORTION EXISTS, SHIFT LEFT TO LINE UP WITH RIGHT
C    PORTION AND 'OR' WITH RIGHT PORTION
C
      J=LAND(IBA(JRE-1),LZBIT(32-NBR))
      K=32-JIM
      I48=LOR(I48,SHFTL(J,K))
      RETURN
C    BIT STRING HAS ONLY ONE BIT
   30 CONTINUE
      I48=0
      JBIND=(JB-1)/32+1
      MSKIND=JB-(JBIND-1)*32
      IF(LAND(MASK(MSKIND),IBA(JBIND)).EQ.MASK(MSKIND)) I48=1
      RETURN
C     END
```

```
      SUBROUTINE ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      IMPLICIT INTEGER(A-Z)
      REAL ESF
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      DIMENSION PELBUF(60), OTBUF(60)
      COMMON/LOGIC/SEARCH,DIAG
      LOGICAL SEARCH,DIAG
C
C*************************** BEGIN PROGRAM *****************************
C
      REWIND PELFIL
      REWIND OTFIL
      ERROR=0
      OTELW=(PELMAX+32-1)/32
      OTLNCT=0
C
C    READ AN ERROR FREE LINE
C
  100 CONTINUE
      READ(PELFIL,END=600,ERR=800) INLNNO,INELCT,PELBUF
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
C
C    READ AN ERROR-CORRUPTED LINE
C
  200 CONTINUE
      READ(OTFIL,END=500,ERR=800) OTLNNO,OTELCT,OTBUF
      OTLNCT=OTLNCT+1
```

UNCLASSIFIED

```
  300 CONTINUE
C
C   COUNT DIFFERENCES BETWEEN TRANSMITTED AND RECEIVED LINES
C
      DO 450 I=1,OTELW
      IF(OTBUF(I).EQ.PELBUF(I)) GO TO 450
      IF(.NOT.DIAG) GO TO 420
      WRITE(TERM,410) INLNNO,OTLNNO,I,PELBUF(I),OTBUF(I)
  410 FORMAT(3I8,2Z12)
  420 CONTINUE
      DO 440 J=1,32
      IF(I48(OTBUF(I),J,1).NE.I48(PELBUF(I),J,1)) ERROR=ERROR+1
  440 CONTINUE
  450 CONTINUE
      IF(OTLNNO-INLNNO) 200,100,580
C
C   ERROR LINE NUMBER GREATER THAN GOOD LINE NUMBER;
C   COUNT DIFFERENCES BETWEEN GOOD AND ALL WHITE LINE
C
  500 CONTINUE
      DO 550 I=1,OTELW
      IF(PELBUF(I).EQ.0) GO TO 550
      IF(.NOT.DIAG) GO TO 520
      WRITE(TERM,410) INLNNO,OTLNNO,I,PELBUF(I),OTBUF(I)
  520 CONTINUE
      DO 540 J=1,32
      IF(I48(PELBUF(I),J,1).NE.0) ERROR=ERROR+1
  540 CONTINUE
  550 CONTINUE
C
  580 READ(PELFIL,END=600,ERR=800) INLNNO,INELCT,PELBUF
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 580
C
      GO TO 300
C
C   CALCULATE ERROR SENSITIVITY FACTOR
C
  600 CONTINUE
      ESF=0.
      IF(ERRCNT.LE.0) GO TO 650
      ESF=FLOAT(ERROR)/FLOAT(ERRCNT)
  650 CONTINUE
C
      WRITE(LPFIL,700) ERROR,ERRCNT,ESF,OTLNCT
  700 FORMAT('0NUMBER OF INCORRECT PELS =',I10/
     *       '0NUMBER OF BITS IN ERROR TRANSMITTED =',I10/
     *       '0ERROR SENSITIVITY FACTOR =',F12.4/
     *       '0TOTAL NUMBER OF OUTPUT LINES PROCESSED = ',I8)
C
      RETURN
  800 CONTINUE
      STOP 800
      END
```

```
      PROGRAM WRITAPE
C
      IMPLICIT INTEGER (A-Z)
      INTEGER PELBUF(60),OTBUF(1500)
      DATA PELMAX,PELFIL,OTFIL,TERM/172?,.,.,6/
C
C ---------------------- BEGIN PROGRAM ----------------------
C
      INLNCT=0
  150 CONTINUE
      DO 1 I=1,60
    1 PELBUF(I)=0
      READ(PELFIL,END=500,ERR=600) INLNNO,INELCT,PELBUF
      INLNCT=INLNCT+1
      IC=I40(PELBUF,1,1)
      POLAR=IC
      J=1
      RUN=0
      DO 200 I=1,PELMAX
      PEL=I40(PELBUF,I,1)
      IF(PEL.EQ.POLAR) GO TO 19
      OTBUF(J)=RUN
      J=J+1
      RUN=1
      POLAR=MOD(POLAR+1,2)
      GO TO 200
   19 CONTINUE
      RUN=RUN+1
  200 CONTINUE
      OTBUF(J)=RUN
      ID=1
      IF=25
      WRITE(OTFIL,300) IC,J
  300 FORMAT(25(I4)
      J1=J
  316 IF(J.GT.25) GO TO 315
      JID1=J+ID-1
      WRITE(OTFIL,300) (OTBUF(K),K=ID,JID1)
      GO TO 400
  315 CONTINUE
      WRITE(OTFIL,300) (OTBUF(K),K=ID,IF)
      ID=IF+1
      IF=IF+25
      J=J-25
      IF(J.EQ.0) GO TO 400
      GO TO 316
  400 CONTINUE
C     WRITE(TERM,410) IC,J1
  410 FORMAT(5X,I4,5X,I6)
C     WRITE(TERM,420) (OTBUF(K),K=1,J1)
  420 FORMAT(2X,2)(I4,2X))
      GO TO 150
  500 CONTINUE
      WRITE(TERM,510) INLNCT,INLNNO
  510 FORMAT(' LINES WRITTEN =',I6,'; LAST LINE NUMBER =',I6)
      STOP
  600 CONTINUE
      STOP 6?
      END
```

```
START OF DCEC UPRINT PROGRAM            DSNAME=D0031.CONVERT.FORT
C       PROGRAM CONVERT
C
C    THIS PROGRAM CONVERTS BINARY FORMAT USED BY COMPRESSION
C    ALGORITHMS TO THE FOLLOWING BINARY FORMAT:
C
C         1728 BITS (216 BYTES) PER RECORD;
C
C    EACH LINE OF 1728 PELS BECOMES ONE RECORD
C
        IMPLICIT INTEGER(A-Z)
        INTEGER PELBUF(60),OTBUF(54)
        EQUIVALENCE (PELBUF(1),OTBUF(1))
        INLNCT=0
  100   READ(1,END=500,ERR=600) INLNNO,INELCT,PELBUF
        INLNCT=INLNCT+1
        WRITE(2,ERR=700) OTBUF
        GO TO 100
C
  500   CONTINUE
        WRITE(5,510) INLNCT,INLNNO
  510   FORMAT(' LINES WRITTEN =',I6,'; LAST LINE NUMBER =',I6)
        STOP
  600   CONTINUE
        STOP 600
  700   STOP 700
        E N D
0            END OF DCEC UPRINT PROGRAM            LINES PRINTED=     26
```

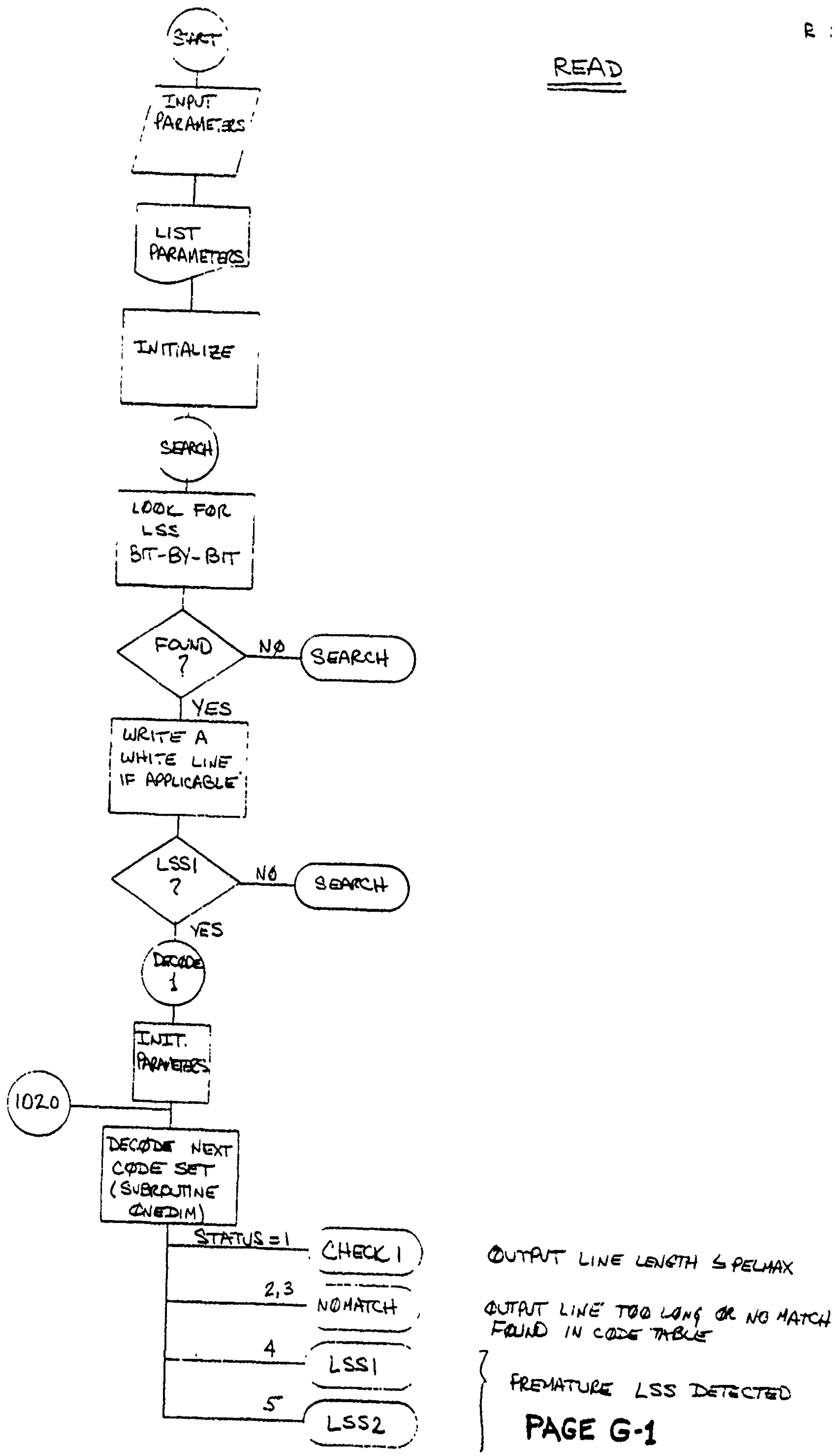
APPENDIX G

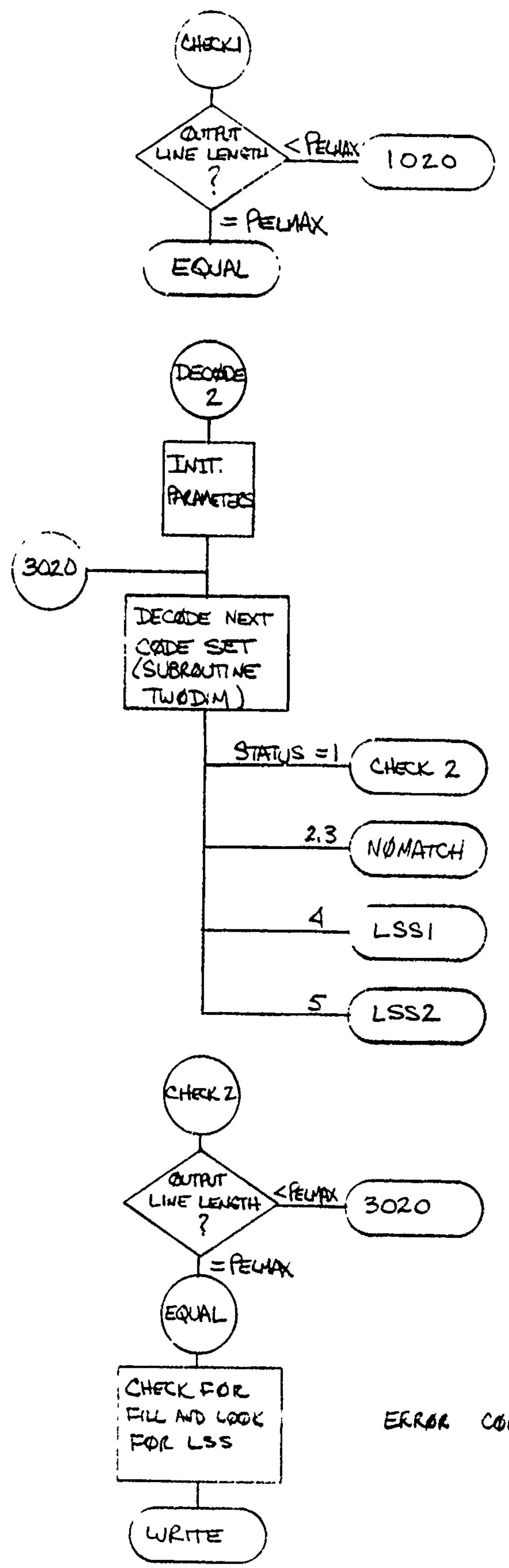
PROGRAM FLOW CHART

FOR JAPAN ALGORITHM

R : 9

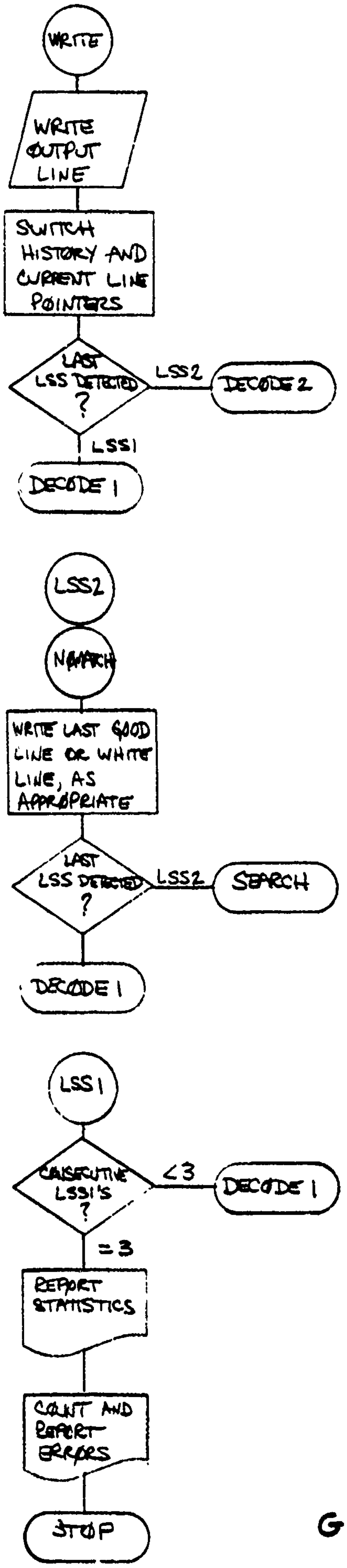
READ


START
INPUT PARAMETERS
LIST PARAMETERS
INITIALIZE
SEARCH
LOOK FOR LSS BIT-BY-BIT
FOUND ?
NO
SEARCH
YES
WRITE A WHITE LINE IF APPLICABLE
LSS1 ?
NO
SEARCH
YES
DECODE 1
INIT. PARAMETERS
1020
DECODE NEXT CODE SET (SUBROUTINE ONEDIM)
STATUS = 1
CHECK 1
2,3
NOMATCH
4
LSS1
5
LSS2
OUTPUT LINE LENGTH ≤ PELMAX
OUTPUT LINE TOO LONG OR NO MATCH FOUND IN CODE TABLE
PREMATURE LSS DETECTED



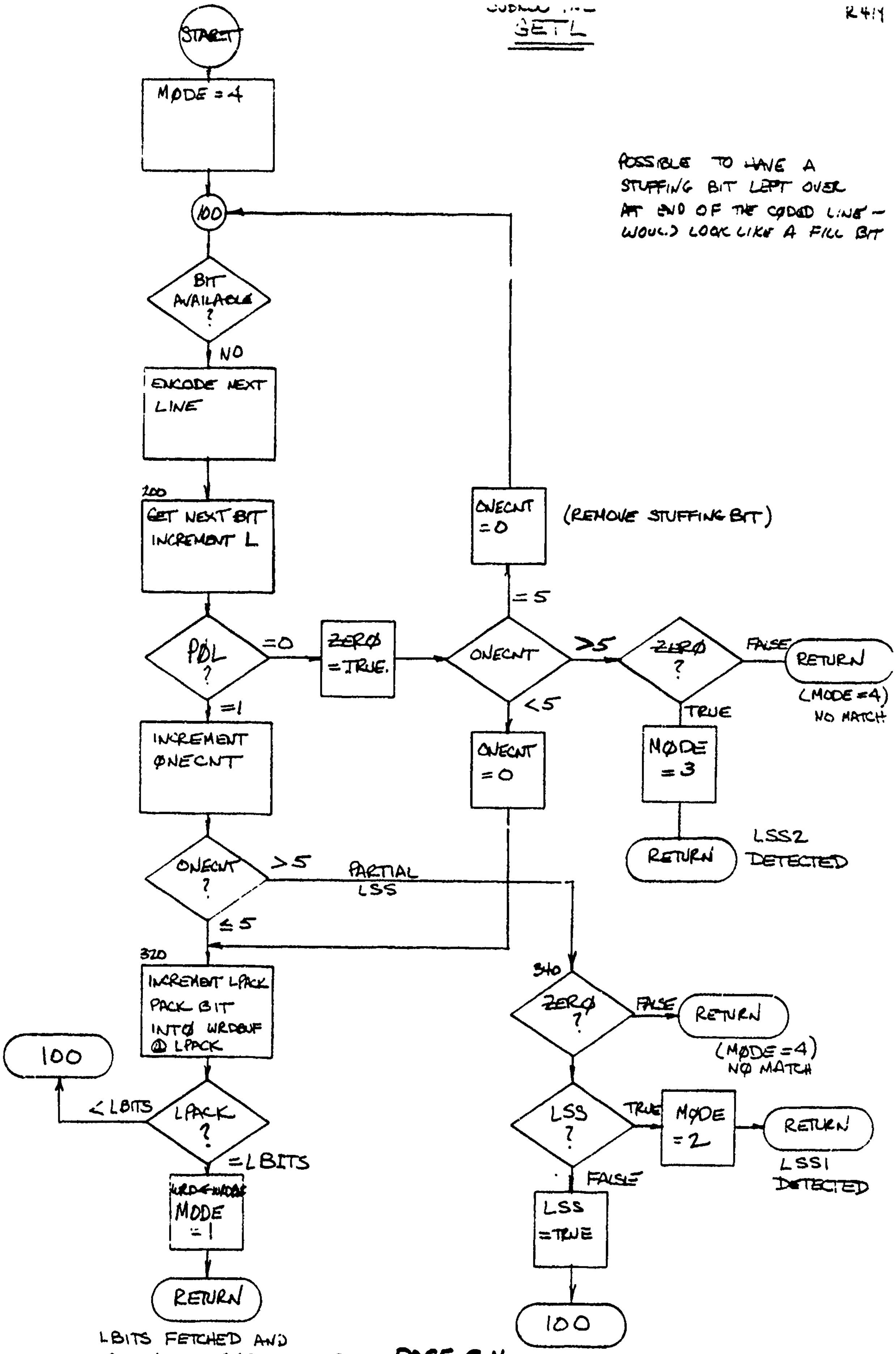
ERROR CONDITION: NEXT NON-ZERO BIT GROUP NOT AN LSS

R 3/9

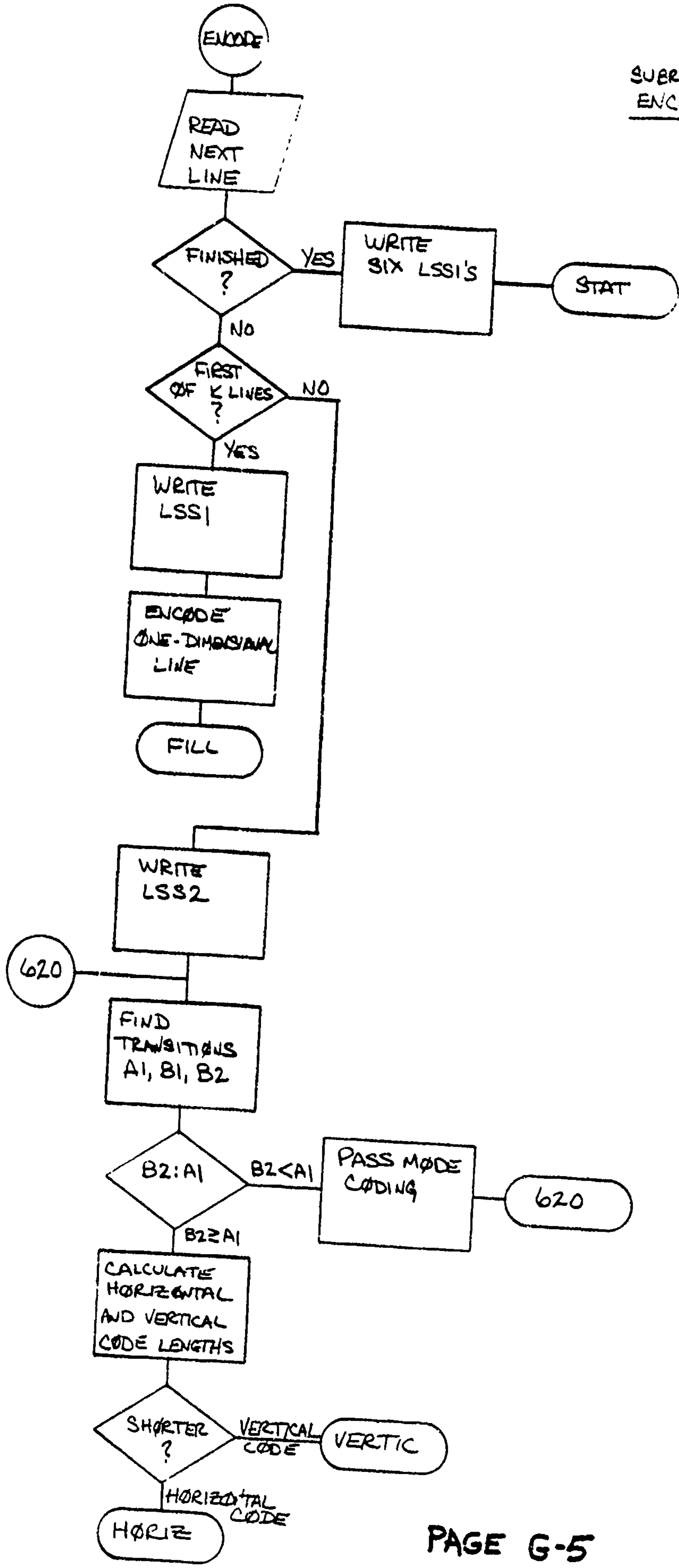

WRITE
WRITE OUTPUT LINE
SWITCH HISTORY AND CURRENT LINE POINTERS
LAST LSS DETECTED ?
LSS2
DECODE 2
LSS1
DECODE 1
LSS2
NMATCH
WRITE LAST GOOD LINE OR WHITE LINE, AS APPROPRIATE
LAST LSS DETECTED ?
LSS2
SEARCH
DECODE 1
LSS1
CONSECUTIVE LSS1'S ?
<3
DECODE 1
=3
REPORT STATISTICS
COUNT AND REPORT ERRORS
STOP

GETL

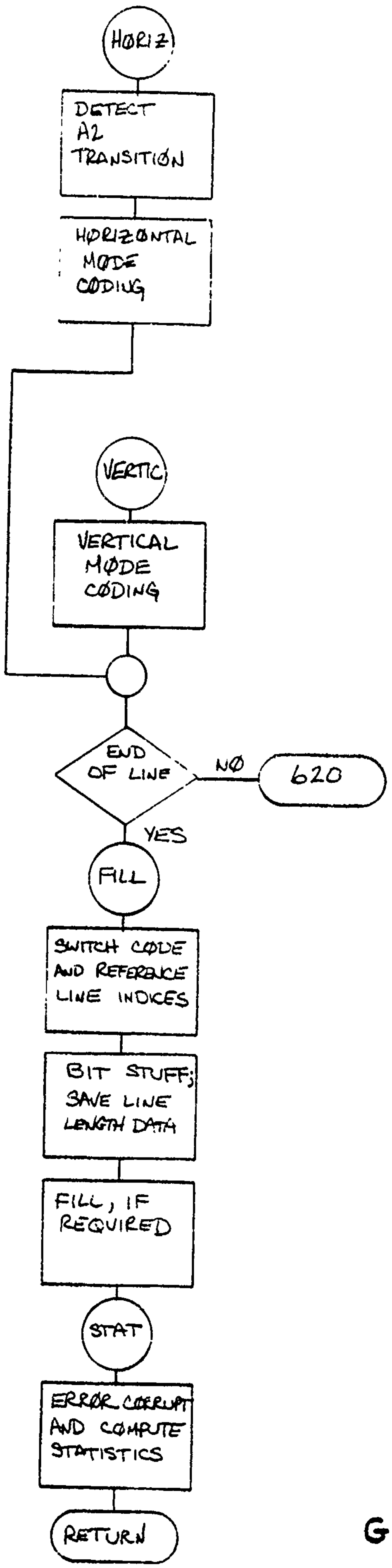
R414


START
MODE = 4
100
BIT AVAILABLE ?
NO
ENCODE NEXT LINE
200
GET NEXT BIT INCREMENT L
POL ?
=0
ZERO = TRUE
ONECNT
=5
ONECNT = 0
(REMOVE STUFFING BIT)
POSSIBLE TO HAVE A STUFFING BIT LEFT OVER AT END OF THE CODED LINE — WOULD LOOK LIKE A FILL BIT
>5
ZERO ?
FALSE
RETURN
(MODE = 4) NO MATCH
TRUE
MODE = 3
RETURN
LSS2 DETECTED
<5
ONECNT = 0
=1
INCREMENT ONECNT
ONECNT ?
>5
PARTIAL LSS
≤5
320
INCREMENT LPACK PACK BIT INTO WRDBUF @ LPACK
340
ZERO ?
FALSE
RETURN
(MODE = 4) NO MATCH
LSS ?
TRUE
MODE = 2
RETURN
LSS1 DETECTED
FALSE
LSS = TRUE
100
100
< LBITS
LPACK ?
= LBITS
MODE = 1
RETURN
LBITS FETCHED AND PUT IN WRD RIGHT JUSTIFIED

SUBROUTINE ENCODR



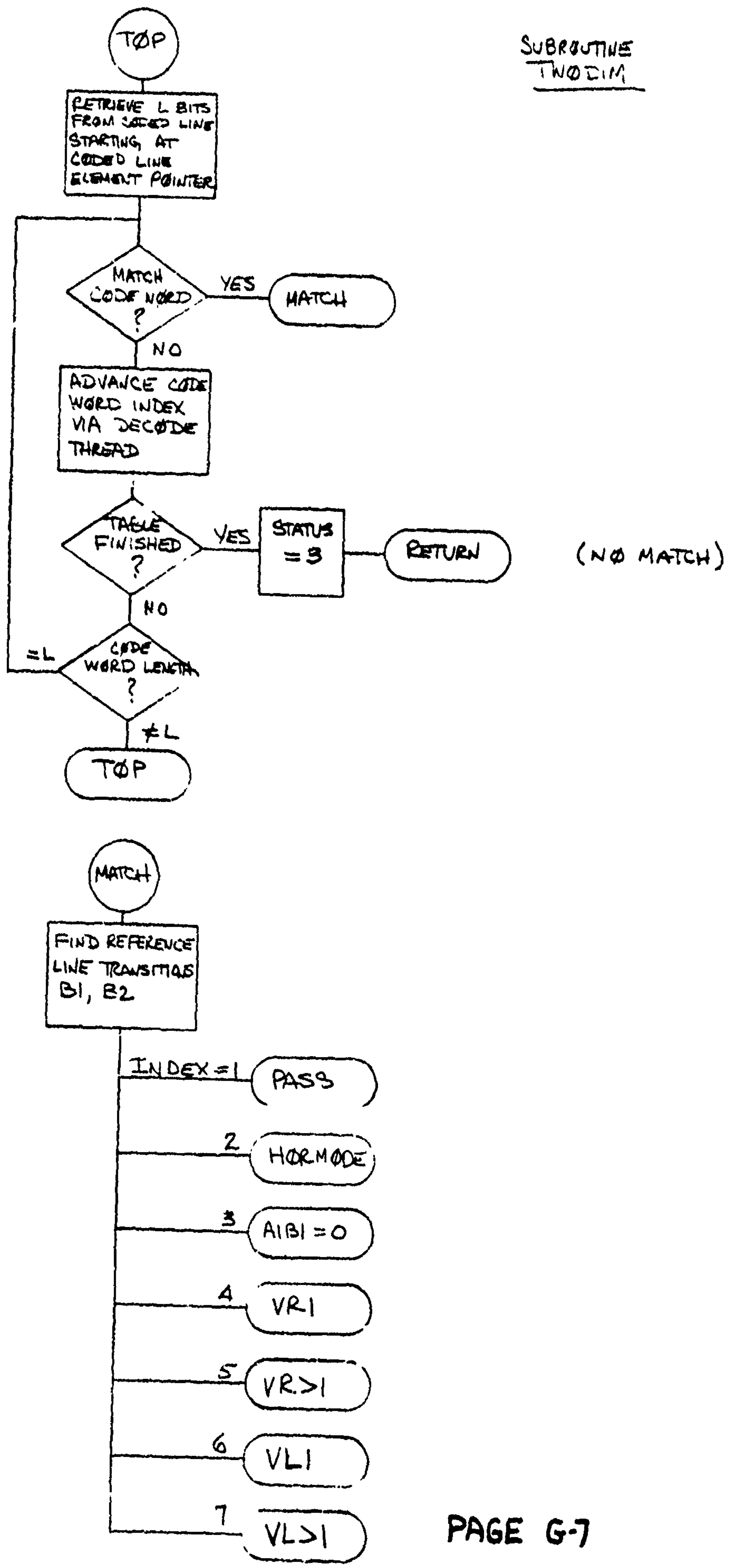
R 6/9



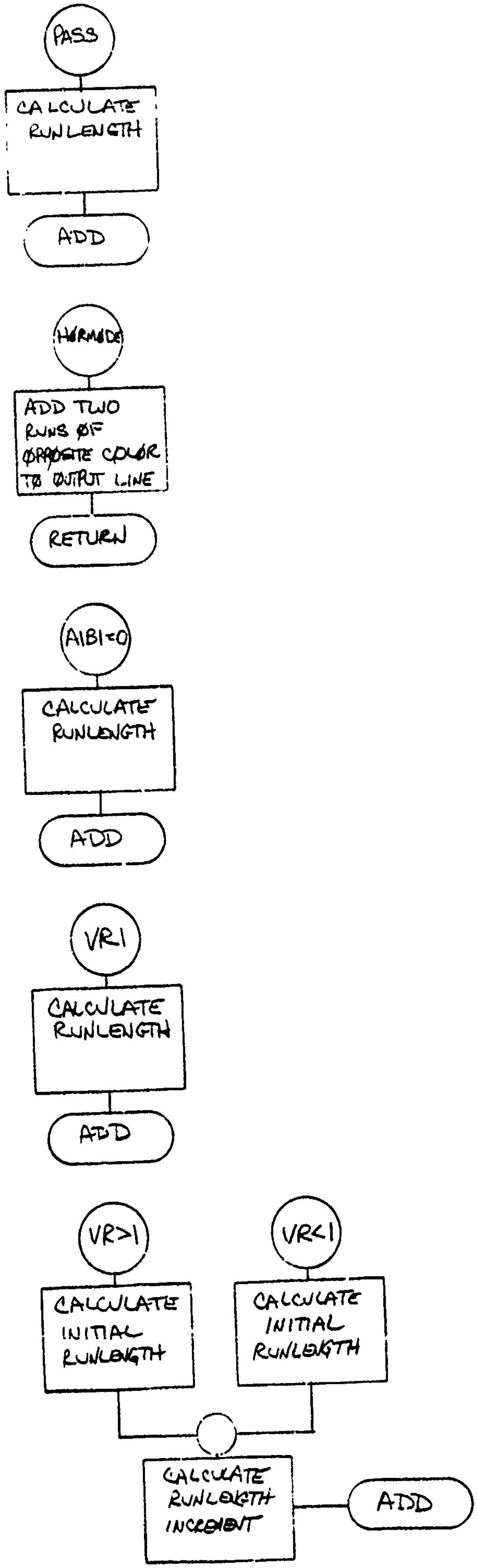
HØRIZ
DETECT
A2
TRANSITIØN
HØRIZØNTAL
MØDE
CØDING
VERTIC
VERTICAL
MØDE
CØDING
END
OF LINE
NØ
620
YES
FILL
SWITCH CØDE
AND REFERENCE
LINE INDICES
BIT STUFF;
SAVE LINE
LENGTH DATA
FILL, IF
REQUIRED
STAT
ERRØR CØRRUPT
AND CØMPUTE
STATISTICS
RETURN

SUBROUTINE
TWODIM

R7/1

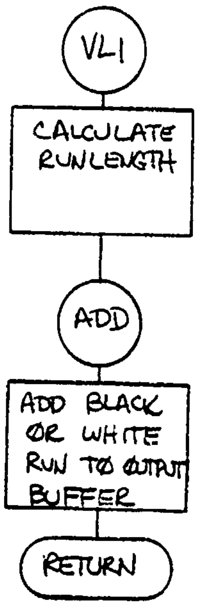


K8/9


PASS
CALCULATE RUNLENGTH
ADD
HORMODE
ADD TWO RUNS OF OPPOSITE COLOR TO OUTPUT LINE
RETURN
A1B1=0
CALCULATE RUNLENGTH
ADD
VR1
CALCULATE RUNLENGTH
ADD
VR>1
VR<1
CALCULATE INITIAL RUNLENGTH
CALCULATE INITIAL RUNLENGTH
CALCULATE RUNLENGTH INCREMENT
ADD

PAGE G-8

R9/9



STATUS = 1: LINE LENGTH $\leq$ PELMAX
STATUS = 2: LINE TØØ LØNG (> PELMAX)
STATUS = 3: NEGATIVE RUNLENGTH

APPENDIX H

COMPUTER PROGRAM CODE LISTING

JAPAN ALGORITHM

START OF DCEC JPRINT PROGRAM   DSNAME=D0031.JPREAD.FORT

```
C     PROGRAM JPREAD
      IMPLICIT INTEGER(A-Z)
      REAL CF3,CF4,ERRATE
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRDBUF,LPACK,
     *            INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
C   READ INPUT PARAMETERS
   90 WRITE(TERM,100)
  100 FORMAT('$PARAMETERS: INPUT(=I), OR DEFAULT(=D)?')
      READ(TERM,110,ERR=90) INSW
  110 FORMAT(A1)
      IF (INSW.EQ.DD) GO TO 315
      IF (INSW.NE.II) GO TO 90
C
C   READ DIAGNOSTIC SWITCH
C
  114 WRITE(TERM,115)
  115 FORMAT('$DIAGNOSTIC PRINTOUT? (Y OR N): ')
      READ(TERM,110) INSW
      IF(INSW.EQ.YY) GO TO 116
      IF(INSW.EQ.NN) GO TO 120
      GO TO 114
  116 CONTINUE
      DIAG=.TRUE.
C
C   READ MAXIMUM NUMBER OF PELS PER LINE
C
  120 CONTINUE
      WRITE(TERM,130)
  130 FORMAT('$ENTER MAXIMUM NUMBER OF PELS PER LINE: ')
      READ(TERM,140,ERR=120) PELMAX
  140 FORMAT(I4)
      IF(PELMAX.GE.1.AND.PELMAX.LE.1728) GO TO 160
      WRITE(TERM,150) PELMAX
  150 FORMAT('0NUMBER OUT OF RANGE (=',I6,')')
      GO TO 120
C
C   READ VERTICAL SAMPLING
C
  160 CONTINUE
      WRITE(TERM,170)
  170 FORMAT('$ENTER VERTICAL SAMPLING: ')
      READ(TERM,180,ERR=160) VRES
  180 FORMAT(I2)
      IF(VRES.GE.1.AND.VRES.LE.10) GO TO 190
      WRITE(TERM,150) VRES
      GO TO 160
C
C   READ PARAMETER K
C
  190 CONTINUE
      WRITE(TERM,192)
  192 FORMAT('$ENTER PARAMETER K: ')
      READ(TERM,140,ERR=190) K
      IF(K.GE.1.AND.K.LE.3000) GO TO 200
      WRITE(TERM,150) K
      GO TO 190
C
C   READ ERROR PATTERN PHASE
C
```

```
  200 CONTINUE
      WRITE(TERM,210)
  210 FORMAT('$ENTER ERROR PATTERN PHASE: ')
      READ(TERM,220,ERR=200) EPHASE
  220 FORMAT(I1)
      IF(EPHASE.GE.0.AND.EPHASE.LE.3) GO TO 240
      WRITE(TERM,150) EPHASE
      GO TO 200
C
C   READ MINIMUM COMPRESSED LINE LENGTH
C
  240 CONTINUE
      WRITE(TERM,250)
  250 FORMAT('$ENTER MINIMUM COMPRESSED LINE LENGTH: ')
      READ(TERM,140,ERR=240) CMPMAX
      IF(CMPMAX.GE.0.AND.CMPMAX.LE.1728) GO TO 320
      WRITE(TERM,150) CMPMAX
      GO TO 240
C
C   READ NUMBER OF SCAN LINES TO BE PROCESSED
  320 CONTINUE
      WRITE(TERM,330)
  330 FORMAT('$NUMBER OF SCAN LINES TO BE PROCESSED=? ')
      READ(TERM,140,ERR=320) LINMAX
      IF(LINMAX.GE.1.AND.LINMAX.LE.3000) GO TO 280
      WRITE(TERM,150) LINMAX
      GO TO 320
C
C   READ ERROR MODE
C
  280 CONTINUE
      WRITE(TERM,290)
  290 FORMAT('$ERROR MODE=? (M=MANUAL,T=TAPE,N=NO ERRORS)')
      READ(TERM,110,ERR=280) ERRMOD
      IF(ERRMOD.EQ.MM) GO TO 300
      IF(ERRMOD.EQ.TT) GO TO 315
      IF(ERRMOD.NE.NN) GO TO 280
      GO TO 350
C
C   READ ERROR LOCATIONS
C
  300 CONTINUE
      ERRLIM=1
  305 READ(TERM,140) ERRORS(ERRLIM)
      IF(ERRORS(ERRLIM).EQ.9999) GO TO 310
      ERRLIM=ERRLIM+1
      GO TO 305
  310 CONTINUE
      ERRLIM=ERRLIM-1
      GO TO 350
C
C   READ ERROR TAPE FILE AND OPEN
C
  315 CONTINUE
C
      ERRLIM=1
      READ(ERFIL,318,END=317) ERRORS(ERRLIM)
      ERRLIM=ERRLIM+1
  316 READ(ERFIL,318,END=317) ERRORS(ERRLIM)
  318 FORMAT(I16)
      ERRORS(ERRLIM)=ERRORS(ERRLIM)+ERRORS(ERRLIM-1)
      ERRLIM=ERRLIM+1
      GO TO 316
  317 ERRLIM=ERRLIM-1
C
  350 CONTINUE
C
  360 CONTINUE
C   WRITE INPUT PARAMETERS
C
      WRITE(LPFIL,400) PELMAX,VRES,K,EPHASE,CMPMAX,LINMAX
  400 FORMAT('1INPUT PARAMETERS:'/
     *       '0MAXIMUM NUMBER OF PELS PER LINE=',I6/
     *       '0VERTICAL SAMPLING: N=',I4/
     *       '0PARAMETER K =',I4/
     *       '0ERROR PATTERN PHASE =',I4/
     *       '0MINIMUM COMPRESSED LINE LENGTH =',I4,' BITS'/
     *       '0NUMBER OF SCAN LINES TO BE PROCESSED =',I6)
      IF(ERRMOD.EQ.NN) WRITE(LPFIL,410)
  410 FORMAT ('0NO ERRORS INSERTED')
      IF(ERRMOD.EQ.MM) WRITE(TERM,140) (ERRORS(I),I=1,ERRLIM)
      IF(ERRMOD.EQ.TT) WRITE(TERM,420) ERRLIM
```

```
  420 FORMAT(I12,' ERRORS OBTAINED FROM ERROR TAPE')
C******************** BEGIN PROGRAM ********************************
C
C    INITIALIZE
C
      TCDEL=0
      TCDATA=0
      ERRPNT=1
      ERRCNT=0
      INLNCT=0
      ERROFF=EPHASE*1024
      CDELCT=32
      OTELP=1
      CDELP=32+1
      CONSEC=1
      INREF=1
      INCOD=2
      OTREF=1
      OTCOD=2
      WRDBUF=0
      LPACK=0
C
      DO 800 I=1,240
      STFBUF(I)=0
      CDBUF(I)=0
  800 CONTINUE
      DO 850 I=1,60
      OTBUF(I,OTREF)=0
      OTBUF(I,OTCOD)=0
      PELBUF(I,INREF)=0
      PELBUF(I,INCOD)=0
  850 CONTINUE
      SEARCH=.TRUE.
      SYNC=.FALSE.
      WRITE=.FALSE.
C
C    SEARCH MODE: LOOK FOR LSS1 BIT-BY-BIT
C
  900 CONTINUE
      L=0
      ZERO=.FALSE.
      LSS=.FALSE.
      ONECNT=0
      WRDBUF=0
      LPACK=0
      CALL GETL(8,MODE,LBITS,L)
C
C     IF(DIAG) WRITE(TERM,140) MODE
C     IF(MODE.NE.2) STOP 900
C
      GO TO (910,930,930,920),MODE
      STOP 900
  910 CONTINUE
C
C    LSS NOT FOUND; ADVANCE POINTER AND TRY AGAIN
C
      CDELP=CDELP+1
      GO TO 900
  920 CONTINUE
C
C    SIX ONES DETECTED WITHOUT AN INITIAL ZERO
C
      IF(DIAG) WRITE (TERM,925) CDELP
  925 FORMAT('0SOMETHING ROTTEN AT CDELP=',I8)
      GO TO 910
  930 CONTINUE
C
C    LSS FOUND
C
      SEARCH=.FALSE.
      CDELP=CDELP+L
      IF(WRITE) GO TO 935
      WRITE=.TRUE.
      GO TO 960
  935 CONTINUE
C
C    SET OUTPUT DECODE LINE TO 0 AND WRITE OUT
      DO 950 I=1,60
      OTBUF(I,OTCOD)=0
  950 CONTINUE
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
```

```
 960   CONTINUE
       IF(MODE-2)965,1000,900
 965   STOP 965
 1000  CONTINUE
C
C    PERFORM ONE-DIMENSIONAL DECODE OF A COMPLETE LINE
C    FIRST,SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
       DO 1010 I=1,60
       OTBUF(I,OTCOD)=0
 1010  CONTINUE
C
       INDEX=3
       COLOR=1
       OTELP=1
C
       ZERO=.FALSE.
       LSS=.FALSE.
       ONECNT=0
 1020  CONTINUE
       CALL ONEDIM(INDEX,COLOR,STATUS,L)
       GO TO (1030,1070,1070,1035,1040),STATUS
C                1    2     3    4    5
       STOP 1000
C
C    RUN ADDED; CHECK LENGTH OF OUTPUT LINE
C
 1030  CONTINUE
       ONE=.TRUE.
       IF(OTELP-1-PELMAX) 1031,1032,1050
 1031  CONTINUE
       IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
       INDEX=3
       GO TO 1020
 3000  CONTINUE
C
C    PERFORM TWO-DIMENSIONAL DECODE
C
C
C    FIRST,SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
       DO 3010 I=1,60
       OTBUF(I,OTCOD)=0
 3010  CONTINUE
C
       INDEX=3
       COLOR=1
       OTELP=1
C
       ZERO=.FALSE.
       LSS=.FALSE.
       ONECNT=0
 3020  CONTINUE
       CALL TWODIM(INDEX,COLOR,STATUS,L)
       GO TO (3030,1070,1070,1035,1040),STATUS
C                1    2     3    4    5
       STOP 3000
C
C    RUN ADDED; LOOK FOR NEXT RUN
C
 3030  CONTINUE
       ONE=.FALSE.
       IF(OTELP-1-PELMAX) 3031,1032,1050
 3031  CONTINUE
       IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
       INDEX=3
       GO TO 3020
C
C
C    LINE LENGTH=PELMAX; CHECK FOR FILL AND LOOK FOR LSS
C
 1032  CONTINUE
       ZERO=.FALSE.
       LSS=.FALSE.
       ONECNT=0
 1033  CONTINUE
       WRDBUF=0
       LPACK=0
       L=0
       CALL GETL(1,MODE,LBITS,L)
```

```
C
      GO TO (1034,1050,1050,1050),MODE
C
C   CHECK FOR FILL
C
 1034 CONTINUE
C
      CDELP=CDELP+L
      IF(LBITS.EQ.0) GO TO 1033
      L=0
      WRDBUF=0
      LPACK=0
      CALL GETL(6,MODE,LBITS,L)
      GO TO (1070,1060,1060,1080),MODE
C
C   PREMATURE LSS DETECTED
C
C
C   LSS1 DETECTED
C
 1035 CONTINUE
      CDELP=CDELP+L
      STATUS=4
      IF(OTELP.LE.5) CONSEC=CONSEC+1
      IF(CONSEC-2)1080,1000,2000
C
C   LSS2 DETECTED
C
 1040 CONTINUE
      CDELP=CDELP+L
      STATUS=5
C
      GO TO 1080
C
C   PROBLEMS,PROBLEMS
C
 1050 STOP 1050
C
C   LINE LENGTH CORRECT, LSS DETECTED PROPERLY; WRITE OUTPUT LINE
C
 1060 CONTINUE
      CDELP=CDELP+L
      WRITE(OTFIL)OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      CTLNNO=LNNOBF
      CONSEC=1
      IF(ONE) SYNC=.TRUE.
      TEMP=OTREF
      OTREF=OTCOD
      OTCOD=TEMP
      IF(MODE.EQ.2) GO TO 1000
      GO TO 3000
C
C   LINE TOO LONG OR NO MATCH
C
 1070 CONTINUE
      WRITE=.FALSE.
C
C   LINE SHORT
C
 1080 CONTINUE
      IF(.NOT.SYNC) GO TO 1090
C
C   WRITE LAST GOOD LINE
C
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTREF),I=1,60)
      SYNC=.FALSE.
      GO TO 1110
 1090 CONTINUE
C
C   WRITE A WHITE LINE
C
      DO 1100 I=1, 60
 1100 OTBUF(I,OTCOD)=0
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
 1110 OTLNNO=LNNOBF
      IF(STATUS.EQ.4) GO TO 1000
      SEARCH=.TRUE.
      GO TO 900
C
C   END OF MESSAGE
C
 2000 CONTINUE
```

```
      WRITE(LPFIL,2010) CONSEC
 2010 FORMAT('0END OF MESSAGE DETECTED (',I2,' EOL''S)')
C
C    REPORT COMPRESSION FACTOR, ERROR SENSITIVITY FACTOR,BIT ERROR RATE
C
      ERRATE=FLOAT(ERRCNT)/FLOAT(TCDEL)
      WRITE(LPFIL,2020) TCDEL,TCDATA,TSTFBT,INLNCT,ERRATE
 2020 FORMAT('0TOTAL NUMBER OF CODED BITS = ',I8/
     *       '0TOTAL NUMBER OF CODED DATA BITS = ',I8/
     *       '0TOTAL NUMBER OF STUFFING BITS = ',I8/
     *       '0TOTAL NUMBER OF INPUT LINES PROCESSED = ',I8/
     *       '0BIT ERROR RATE = ',G14.6)
C
      CALL STATS(STAT,INLNCT,DIAG)
      CF3=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDEL)
      CF4=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDATA)
C
      WRITE(LPFIL,2030) CF3,CF4
 2030 FORMAT('0COMPRESSION FACTOR FOR G3 MACHINE (CF3) =',F8.4/
     *       '0COMPRESSION FACTOR FOR G4 MACHINE (CF4) =',F8.4)
C
      CALL ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      STOP
      END
      SUBROUTINE GETL(LBITS,MODE,WRD,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRDBUF,LPACK,
     *            INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C*************************** BEGIN PROGRAM ***************************
C
      MODE=4
C
C    RETRIEVE NEXT BIT FROM CDBUF
C
 100  CONTINUE
C
C    ENCODE A NEW LINE IF NECESSARY
C
      IF(L+CDELP.LE.CDELCT) GO TO 200
      IF(CDELCT-CDELP+1) 170,190,180
  170 STOP 170
  180 CONTINUE
      STFBUF(1)=I48(STFBUF,CDELP,CDELCT-CDELP+1)
  190 CONTINUE
      CDELP=32-(CDELCT-CDELP)
      CALL ENCODR
 200  CONTINUE
      POL=I48(STFBUF,CDELP+L,1)
      L=L+1
      IF(POL)220,240,300
 220  STOP 220
 240  ZERO=.TRUE.
      IF(ONECNT-5)310,260,280
 260  ONECNT=0
      GO TO 100
 280  IF(.NOT.ZERO) RETURN
      MODE=3
      RETURN
 300  ONECNT=ONECNT+1
      IF(ONECNT-5)320,320,340
  310 CONTINUE
      ONECNT=0
 320  CONTINUE
      LPACK=LPACK+1
```

```
      IF(POL) 324,330,325
  324 STOP 324
  325 CONTINUE
      CALL MI2B(POL,WRDBUF,LPACK,1)
  330 CONTINUE
      IF(LPACK.LT.LBITS) GO TO 100
      WRD=I4B(WRDBUF,1,LPACK)
      MODE=1
      RETURN
 340  IF(.NOT.ZERO) RETURN
      IF(LSS) GO TO 360
      LSS=.TRUE.
      GO TO 100
 360  MODE=2
      RETURN
      END
      SUBROUTINE ENCODR
C
      IMPLICIT INTEGER(A-Z)
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CCDERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C************************ FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRDBUF,LPACK,
     *            INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
C********************************** BEGIN PROGRAM *******************
C
C    INITIALIZE VARIABLES
C
      CDELCT=32
      CDDATA=0
      DO 50 I=2,240
      CDBUF(I)=0
  50  CONTINUE
C
C    READ INPUT PICTURE FILE
C
  100 CONTINUE
      READ(PELFIL,END=120,ERR=500)
     * INLNNO,INELCT,(PELBUF(I,INCOD),I=1,60)
      IF(MOD(INLNNO,100).EQ.0) WRITE(TERM,110) INLNNO
  110 FORMAT(' INPUT LINE NO. =',I6)
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
      IF(INELCT.LT.PELMAX) CALL EXIT
      INLNCT=INLNCT+1
C
C    LOAD OUTPUT LINE NUMBER BUFFER
C
      LNNOBF=INLNNO
      IF(SEARCH)OTLNNO=LNNOBF
C
      IF(INLNNO.LE.LINMAX) GO TO 140
C
C    WRITE SIX LSS1'S
C
  120 CONTINUE
      DO 130 I=1,6
      CALL CODEVH(8,0,0,0,0,CDELCT,CDDATA)
  130 CONTINUE
      DO 135 I=1,6
      STFBUF(I)=CDBUF(I)
  135 CONTINUE
      GO TO 400
```

```
C
C   FIRST OF K LINES?
C
 140 CONTINUE
      IF(MOD(INLNCT-1,K).NE.0) GO TO 600
C
C   ONE-DIMENSIONAL CODING
C   WRITE ONE LSS1
C
      CALL CODEVH(8,0,0,0,0,CDELCT,CDDATA)
C
      POLAR=1
C
C   TEST COLOR OF FIRST ELEMENT
C
      IF(I48(PELBUF(1,INCOD),1,1).EQ.0) GO TO 150
C
C   FIRST ELEMENT BLACK; ENCODE 0-LENGTH WHITE RUN
C
      CALL CODELN(0,1,CDELCT,CDDATA)
      POLAR=2
C
C   CALCULATE RUN LENGTH AND ENCODE
C
 150 CONTINUE
      RUN=0
      DO 200 I=1,PELMAX
      PEL=I48(PELBUF(1,INCOD),I,1)+1
      IF(PEL.EQ.POLAR) GO TO 180
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 170
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
 160 FORMAT(4I8)
 170 CONTINUE
      RUN=1
      POLAR=MOD(POLAR+2,2)+1
      GO TO 200
 180 CONTINUE
      RUN=RUN+1
 200 CONTINUE
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 210
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
      GO TO 210
C
C   TWO-DIMENSIONAL CODING
C
 600 CONTINUE
C
C   WRITE ONE LSS2
C
      CALL CODEVH(9,0,0,0,0,CDELCT,CDDATA)
C
C   SET A0 TO LEFT EDGE-1 AND POLARITY=WHITE
C
      A0=0
      POL=0
      LEFT=.TRUE.
C
C   DETECT A1
C
 620 CONTINUE
      I=A0+1
      IF(I.GT.PELMAX) GO TO 640
 630 CONTINUE
      PEL=I48(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 640
      I=I+1
      IF(I.LE.PELMAX) GO TO 630
 640 CONTINUE
      A1=I
C
C   DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 665
      PELM1=I48(PELBUF(1,INREF),A0,1)
      IF(LEFT) PELM1=0
 650 CONTINUE
      PEL=I48(PELBUF(1,INREF),I,1)
      IF(PEL.NE.PELM1) GO TO 670
 660 CONTINUE
```

```
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 650
 665  CONTINUE
      B1=I
      GO TO 710
 670  CONTINUE
      IF(PEL.NE.POL) GO TO 690
      GO TO 660
 690  CONTINUE
      B1=I
      POL=PEL
C
C   DETECT B2
C
      I=B1+1
      IF(I.GT.PELMAX) GO TO 710
 700  CONTINUE
      PEL=I4B(PELBUF(1,INREF),I,1)
      IF(PEL.NE.POL) GO TO 720
      I=I+1
      IF(I.LE.PELMAX) GO TO 700
 710  CONTINUE
      B2=I
      GO TO 730
 720  CONTINUE
      B2=I
      POL=PEL
 730  CONTINUE
      IF(.NOT.LEFT) POLAR=I4B(PELBUF(1,INCOD),A0,1)+1
      IF(.NOT.LEFT) GO TO 740
      POLAR=1
      A0=1
      LEFT=.FALSE.
 740  CONTINUE
C
C   TEST FOR PASS MODE
C
      IF(B2.GE.A1) GO TO 750
C
C   PASS MODE CODING (CAN'T END A LINE IN PASS MODE;NEW A0 MUST HAVE
C                     SAME POLARITY AS B2)
C
      CALL CODEVH(1,0,0,0,0,CDELCT,CDDATA)
      A0=B2
      GO TO 620
C
C   CALCULATE LENGTH OF VERTICAL AND HORIZONTAL MODES
C   (EFFECT OF ZERO INSERTION IS IGNORED)
C
 750  CONTINUE
C
C   DO HORIZONTAL FIRST
C
      A1MA0=A1-A0
      HORIZ=0
      IF(A1MA0.LE.63) GO TO 755
      HORIZ=CODE(1,A1MA0/64+64,POLAR)
 755  CONTINUE
      TEMP=MOD(A1MA0,64)+1
      HORIZ=HORIZ+CODE(1,TEMP,POLAR)+4
C
C   CALCULATE VERTICAL LENGTH
C
      MAB=IABS(A1-B1)
      IF(MAB-1) 760,770,780
C
 760  VERTIC=1
      GO TO 790
 770  VERTIC=3
      GO TO 790
 780  VERTIC=MAB+3
 790  CONTINUE
      IF(HORIZ.GT.VERTIC) GO TO 835
C
C   CODE BY HORIZONTAL MODE;FIRST DETECT A2
C
      I=A1+1
      IF(I.GT.PELMAX) GO TO 810
C
C   CALCULATE POLARITY OF A1
C
```

```
      POL=I4B(PELBUF(1,INCOD),A1,1)
  800 CONTINUE
      PEL=I4B(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 820
      I=I+1
      IF(I.LE.PELMAX) GO TO 800
  810 A2=PELMAX+1
      GO TO 830
  820 CONTINUE
      A2=I
  830 CONTINUE
      CALL CODEVH(2,POLAR,A0,A1,A2,CDELCT,CDDATA)
      A0=A2
      GO TO 960
C
C   CODE BY VERTICAL MODE
C
  835 CONTINUE
      IF(VERTIC-3) 840,850,890
C
  840 CALL CODEVH(3,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  850 IF(A1-B1) 870,860,880
C
  860 STOP 860
  870 CALL CODEVH(6,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  880 CALL CODEVH(4,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  890 IF(A1-B1) 910,900,920
C
  900 STOP 900
  910 CALL CODEVH(7,0,A1,B1,0,CDELCT,CDDATA)
      GO TO 950
  920 CALL CODEVH(5,0,A1,B1,0,CDELCT,CDDATA)
  950 CONTINUE
      A0=A1
C
C   TEST FOR END OF LINE
C
  960 CONTINUE
      IF(A0.GT.PELMAX) GO TO 210
      POL=I4B(PELBUF(1,INCOD),A0,1)
      GO TO 620
  210 CONTINUE
C
C      SWITCH CODE & REFERENCE LINES
C
      TEMP=INREF
      INREF=INCOD
      INCOD=TEMP
C
C   BIT STUFFING (ZERO INSERTION)
C
      CALL STUFF(CDBUF,STFBUF,STFBIT,CDELCT)
C
C   SAVE LINE LENGTH(DATA BITS ONLY)
C
      STAT(INLNCT)=CDDATA+8
C
C   CHECK CODED LINE LENGTH
C
      FILL=CMPMAX-(CDELCT-32)
      IF(FILL) 400,400,250
C
C   CODE LINE TOO SHORT; FILL IT TO CMPMAX
  250 CONTINUE
      CDELCT=CDELCT+FILL
C
C   ACCUMULATE STATISTICS AND ERROR CORRUPT
C
  400 CONTINUE
      IF(ERRMOD.EQ.NN) GO TO 390
C
C   ERROR CORRUPT
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C   ERROR IN RANGE OF CODED LINE; CHANGE APPROPRIATE BIT
```

```
C
      BIT=I4B(STFBUF,ERRBIT+32,1)
      BIT=MOD(BIT+1,2)
      CALL MI2B(BIT,STFBUF,ERRBIT+32,1)
      ERRCNT=ERRCNT+1
C
C   INCREMENT ERROR LIST POINTER
C
  360 CONTINUE
      ERRPNT=ERRPNT+1
      IF(ERRPNT.LE.ERRLIM) GO TO 350
C
C   ERROR LIST EXHAUSTED
C
      ERRPNT=ERRPNT-1
      WRITE(LPFIL,370) ERRPNT,ERRORS(ERRPNT)
  370 FORMAT('0ERROR LIST EXHAUSTED AT',I10,'TH ERROR;'/
     *       ' LAST ERROR OCCURRED AT',I10,' BITS')
      ERRMOD=NN
C
C   COMPUTE STATISTICS
C
  390 CONTINUE
      TCDEL=TCDEL+CDELCT-32
      TCDATA=TCDATA+CDDATA
      IF(DIAG) WRITE(TERM,160) INLNCT, CDDATA
      TSTFBT=TSTFBT+STFBIT
C
      IF (.NOT.DIAG) GO TO 460
      CDELW=(CDELCT+32-1)/32
      WRITE(LPFIL,450) (CDBUF(I),I=1,CDELW)
      WRITE(LPFIL,450) (STFBUF(I),I=1,CDELW)
      WRITE(LPFIL,445) STFBIT
 445  FORMAT(I8,'ZEROES INSERTED')
  450 FORMAT(6Z12)
  460 CONTINUE
      RETURN
C
  500 CONTINUE
      CALL EXIT
C
      E N D
      SUBROUTINE CODEVH(MODE,POLAR,A,B,C,CDELCT,CDDATA)
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
      GO TO (100,200,100,100,500,100,500,800,800) ,MODE
C
C   MODE       1   2   3   4   5   6   7   8   9
C
      STOP 129
C
C   PASS MODE(1),VERTICAL MODE: A1B1=0(3), A1B1=1(4,6)
C
 100  CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      CDDATA=CDDATA+CODERD(1,MODE)
      RETURN
C
C   HORIZONTAL MODE(2)
C
  200 CONTINUE
      CALL MI2B(CODERD(3,2),CDBUF,CDELCT+1,CODERD(1,2))
      CDELCT=CDELCT+CODERD(1,2)
      CDDATA=CDDATA+CODERD(1,2)
      CALL CODELN(B-A,POLAR,CDELCT,CDDATA)
      NEWPOL=MOD(POLAR+2,2)+1
      CALL CODELN(C-B,NEWPOL,CDELCT,CDDATA)
      RETURN
C
C   VERTICAL MODE: A1B1>1 (5,7)
C
 500  CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      CDDATA=CDDATA+CODERD(1,MODE)
      LIM=IABS(A-B)-2
      IF(LIM) 530,560,540
  530 STOP 530
```

```
      POL=I4B(PELBUF(1,INCOD),A1,1)
  800 CONTINUE
      PEL=I4B(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 820
      I=I+1
      IF(I.LE.PELMAX) GO TO 800
  810 A2=PELMAX+1
      GO TO 830
  820 CONTINUE
      A2=I
  830 CONTINUE
      CALL CODEVH(2,POLAR,A0,A1,A2,CDELCT,CDDATA)
      A0=A2
      GO TO 960
C
C   CODE BY VERTICAL MODE
C
  835 CONTINUE
      IF(VERTIC-3) 840,850,890
C
  840 CALL CODEVH(3,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  850 IF(A1-B1) 870,860,880
C
  860 STOP 860
  870 CALL CODEVH(6,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  880 CALL CODEVH(4,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  890 IF(A1-B1) 910,900,920
C
  900 STOP 900
  910 CALL CODEVH(7,0,A1,B1,0,CDELCT,CDDATA)
      GO TO 950
  920 CALL CODEVH(5,0,A1,B1,0,CDELCT,CDDATA)
  950 CONTINUE
      A0=A1
C
C   TEST FOR END OF LINE
C
  960 CONTINUE
      IF(A0.GT.PELMAX) GO TO 210
      POL=I4B(PELBUF(1,INCOD),A0,1)
      GO TO 620
  210 CONTINUE
C
C      SWITCH CODE & REFERENCE LINES
C
      TEMP=INREF
      INREF=INCOD
      INCOD=TEMP
C
C   BIT STUFFING (ZERO INSERTION)
C
      CALL STUFF(CDBUF,STFBUF,STFBIT,CDELCT)
C
C   SAVE LINE LENGTH(DATA BITS ONLY)
C
      STAT(INLNCT)=CDDATA*8
C
C   CHECK CODED LINE LENGTH
C
      FILL=CMPMAX-(CDELCT-32)
      IF(FILL) 400,400,250
C
C   CODE LINE TOO SHORT; FILL IT TO CMPMAX
  250 CONTINUE
      CDELCT=CDELCT+FILL
C
C   ACCUMULATE STATISTICS AND ERROR CORRUPT
C
  400 CONTINUE
      IF(ERRMOD.EQ.NN) GO TO 390
C
C   ERROR CORRUPT
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C   ERROR IN RANGE OF CODED LINE; CHANGE APPROPRIATE BIT
```

```
  540 CONTINUE
      DO 550 I=1,LIM
C     CALL MI2B(0,CDBUF,CDELCT+1,1) CDBUF MUST BE INIT. TO 0
      CDELCT=CDELCT+1
      CDDATA=CDDATA+1
 550  CONTINUE
 560  CALL MI2B(1,CDBUF,CDELCT+1,1)
      CDELCT=CDELCT+1
      CDDATA=CDDATA+1
      RETURN
C
C   ADD LSS1 OR LSS2 TO LINE (8,9)
C
 800  CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      RETURN
      END
      SUBROUTINE ONEDIM(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/IN_NNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRDBUF,LPACK,
     *            INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
C   BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
      L=0
      WRDBUF=0
      LPACK=0
 1002 LENBIT=CODE(1,INDEX,COLOR)
      CALL GETL(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z8,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODE(3,INDEX,COLOR)) GO TO 1100
C
C   NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
       INDEX=CODE(2,INDEX,COLOR)
       IF(INDEX.GE.93) GO TO 1190
       IF(CODE(1,INDEX,COLOR).EQ.LENBIT) GO TO 1040
C
C   CODE WORD LONGER; FROM THE TOP
C
       GO TO 1002
C
C   MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C   NOT AN LSS
C
C
C   TEST FOR MAKE UP OR TERMINATING CODE
C
       RUNLEN=INDEX-1
       IF(INDEX.GE.65) RUNLEN=(INDEX-64)*64
       IF(RUNLEN.EQ.0) GO TO 1160
```

```
      IF(COLOR.EQ.1) GO TO 1155
      IF(RUNLEN.LT.0) STOP 1100
C
C   ADD BLACK RUN TO OUTPUT BUFFER
C
      DO 1150 I=1,RUNLEN
      CALL MI28(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C   ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C OUTPUT LINE LESS THAN OR EQUAL TO MAX SPECIFIED
C
 1160 CONTINUE
      IF(INDEX.LT.65) GO TO 1170
      INDEX=3
      GO TO 1000
C
C   RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1170 CONTINUE
      CHCOL=.TRUE.
      STATUS=1
      RETURN
C
C   RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C   NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C   LSS1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C   LSS2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      SUBROUTINE TWODIM(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *           STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C************************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *           CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *           ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRDBUF,LPACK,
     *           INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
```

```
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
C     BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
      L=0
      WRDBUF=0
      LPACK=0
 1002 LENBIT=CODERD(1,INDEX)
      CALL GETL(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z8,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODERD(3,INDEX)) GO TO 1100
C
C     NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODERD(2,INDEX)
      IF(INDEX.GE.8) GO TO 1190
      IF(CODERD(1,INDEX).EQ.LENBIT) GO TO 1040
C
C     CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C     MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C     NOT AN LSS
C
C
C     FIND B1 AND B2
C
      A0=OTELP
      IF(OTELP.EQ.1) A0=0
      POL=COLOR-1
C
C     DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 65
      PELM1=0
      IF(A0.EQ.0) GO TO 50
      PELM1=I4B(OTBUF(1,OTREF),A0,1)
   50 CONTINUE
      PEL=I4B(OTBUF(1,OTREF),I,1)
      IF(PEL.NE.PELM1) GO TO 70
   60 CONTINUE
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 50
   65 CONTINUE
      B1=I
      GO TO 92
   70 CONTINUE
      IF(PEL.NE.POL) GO TO 90
      GO TO 60
   90 CONTINUE
      B1=I
      POL=PEL
C
C     DETECT B2
C
      I=B1+1
      IF(I.GT.PELMAX) GO TO 92
   91 CONTINUE
      PEL=I4B(OTBUF(1,OTREF),I,1)
      IF(PEL.NE.POL) GO TO 92
      I=I+1
      IF(I.LE.PELMAX) GO TO 91
   92 CONTINUE
      B2=I
      GO TO (100,200,300,400,500,600,700),INDEX
      STOP 100
C
C     PASS MODE
C
  100 CONTINUE
```

```
      RUN.EN=B2-OTELP
      CHCOL=.FALSE.
      GO TO (1155,1145),COLOR
C
C     HORIZONTAL MODE
C
 200  CONTINUE
      ENTRY=3
      CALL ONEDIM(ENTRY,COLOR,STATE,L)
      GO TO (210,1180,1190,1200,1205),STATE
 210  CONTINUE
      COLOR=MOD(COLOR+2,2)+1
      ENTRY=3
      CALL ONEDIM(ENTRY,COLOR,STATE,L)
      GO TO (220,1180,1190,1200,1205),STATE
 220  CONTINUE
      CHCOL=.TRUE.
      GO TO 1160
C
C     VERTICAL MODE A1B1=0
C
 300  CONTINUE
      RUNLEN=B1-OTELP
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     VERTICAL MODE VR1 A1B1=1
C
 400  CONTINUE
      RUNLEN=B1-OTELP+1
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     VERTICAL MODE RIGHT A1B1>1
C
 500  CONTINUE
      RUNLEN=B1-OTELP+2
      DIRECT=1
 510  CONTINUE
      L=0
      WRDBUF=0
      LPACK=0
      CALL GETL(1,VMODE,LBITS,L)
      GO TO (520,1200,1205,1190),VMODE
 520  CONTINUE
      CDELP=CDELP+1
      IF(LBITS) 525,530,540
 525  STOP 525
 530  CONTINUE
      RUNLEN=RUNLEN+(1*DIRECT)
      GO TO 510
 540  CONTINUE
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     VERTICAL MODE LEFT VL1 A1B1=1
C
 600  CONTINUE
      RUNLEN=B1-OTELP-1
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     VERTICAL MODE LEFT A1B1>1
C
 700  CONTINUE
      RUNLEN=B1-OTELP-2
      DIRECT=-1
      GO TO 510
C
C     ADD BLACK RUN TO OUTPUT BUFFER
C
 1145 CONTINUE
      IF(RUNLEN) 1190,1160,1147
 1147 CONTINUE
      DO 1150 I=1,RUNLEN
      CALL MI2B(COLOR-1,OTBUF(1,OTCOD),CTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C     ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
```

```
C
 1155 CONTINUE
      IF(RUNLEN.LT.0) GO TO 1190
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C   RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1160 CONTINUE
      STATUS=1
      RETURN
C
C   RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z12)
      STATUS=2
      RETURN
C
C   NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C   LSS1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C   LSS2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      END
      SUBROUTINE STUFF(CDBUF,STFBUF,STFBIT,CDELCT)
      IMPLICIT INTEGER(A-Z)
      DIMENSION CDBUF(240),STFBUF(240)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
C
C   INITIALIZE STFBUF TO 0
C
      DO 50 I=2,240
      STFBUF(I)=0
   50 CONTINUE
      L1CNT=0
      I=32+1+8
      J=I
      STFBUF(1)=CDBUF(1)
C
C   PICK UP LSS
C
      LSS=I4B(CDBUF(2),1,8)
      CALL MI2B(LSS,STFBUF(2),1,8)
  100 CONTINUE
      POL=I4B(CDBUF,I,1)
      IF(POL.EQ.1) GO TO 110
      L1CNT=0
      GO TO 150
  110 L1CNT=L1CNT+1
      CALL MI2B(POL,STFBUF,J,1)
  150 CONTINUE
      I=I+1
      J=J+1
      IF(L1CNT.LE.4) GO TO 200
C     CALL MI2B(0,STFBUF,J,1) NOT NECESSARY
      L1CNT=0
      J=J+1
C
C   TEST IF FINISHED
C
  200 CONTINUE
      IF(I.LE.CDELCT) GO TO 100
      STFBIT=J-1-CDELCT
      CDELCT=J-1
```

```
      RETURN
      END
      BLOCK DATA
C
      IMPLICIT INTEGER(A-Z)
C************************ FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *              STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************* LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,CTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,CNECNT,LNNOBF,WRDBUF,LPACK,
     *              INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
      DATA TERM,LPFIL,PELFIL,OTFIL,ERFIL/5,6,1,2,3/
      DATA DD,II,MM,TT,NN,YY/'D','I','M','T','N','Y'/
      DATA PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX/1728,2,0,96,'T',3000
      DATA K/2/
      DATA DIAG/.FALSE./
C
       DATA CODE(1,  1,1),CODE(2,  1,1),CODE(3,  1,1)/ 8, 70,Z0035/
       DATA CODE(1,  2,1),CODE(2,  2,1),CODE(3,  2,1)/ 6, 90,Z0007/
       DATA CODE(1,  3,1),CODE(2,  3,1),CODE(3,  3,1)/ 4,  4,Z0007/
       DATA CODE(1,  4,1),CODE(2,  4,1),CODE(3,  4,1)/ 4,  5,Z0008/
       DATA CODE(1,  5,1),CODE(2,  5,1),CODE(3,  5,1)/ 4,  6,Z000B/
       DATA CODE(1,  6,1),CODE(2,  6,1),CODE(3,  6,1)/ 4,  7,Z000C/
       DATA CODE(1,  7,1),CODE(2,  7,1),CODE(3,  7,1)/ 4,  8,Z000E/
       DATA CODE(1,  8,1),CODE(2,  8,1),CODE(3,  8,1)/ 4,  9,Z000F/
       DATA CODE(1,  9,1),CODE(2,  9,1),CODE(3,  9,1)/ 5, 10,Z0013/
       DATA CODE(1, 10,1),CODE(2, 10,1),CODE(3, 10,1)/ 5, 11,Z0014/
       DATA CODE(1, 11,1),CODE(2, 11,1),CODE(3, 11,1)/ 5, 12,Z0007/
       DATA CODE(1, 12,1),CODE(2, 12,1),CODE(3, 12,1)/ 5, 65,Z0008/
       DATA CODE(1, 13,1),CODE(2, 13,1),CODE(3, 13,1)/ 6, 14,Z0008/
       DATA CODE(1, 14,1),CODE(2, 14,1),CODE(3, 14,1)/ 6, 15,Z0003/
       DATA CODE(1, 15,1),CODE(2, 15,1),CODE(3, 15,1)/ 6, 16,Z0034/
       DATA CODE(1, 16,1),CODE(2, 16,1),CODE(3, 16,1)/ 6, 17,Z0035/
       DATA CODE(1, 17,1),CODE(2, 17,1),CODE(3, 17,1)/ 6, 18,Z002A/
       DATA CODE(1, 18,1),CODE(2, 18,1),CODE(3, 18,1)/ 6, 19,Z002B/
       DATA CODE(1, 19,1),CODE(2, 19,1),CODE(3, 19,1)/ 7, 20,Z0027/
       DATA CODE(1, 20,1),CODE(2, 20,1),CODE(3, 20,1)/ 7, 21,Z000C/
       DATA CODE(1, 21,1),CODE(2, 21,1),CODE(3, 21,1)/ 7, 22,Z0008/
       DATA CODE(1, 22,1),CODE(2, 22,1),CODE(3, 22,1)/ 7, 23,Z0017/
       DATA CODE(1, 23,1),CODE(2, 23,1),CODE(3, 23,1)/ 7, 24,Z0003/
       DATA CODE(1, 24,1),CODE(2, 24,1),CODE(3, 24,1)/ 7, 25,Z0004/
       DATA CODE(1, 25,1),CODE(2, 25,1),CODE(3, 25,1)/ 7, 26,Z0028/
       DATA CODE(1, 26,1),CODE(2, 26,1),CODE(3, 26,1)/ 7, 27,Z002B/
       DATA CODE(1, 27,1),CODE(2, 27,1),CODE(3, 27,1)/ 7, 28,Z0013/
       DATA CODE(1, 28,1),CODE(2, 28,1),CODE(3, 28,1)/ 7, 29,Z0024/
       DATA CODE(1, 29,1),CODE(2, 29,1),CODE(3, 29,1)/ 7, 68,Z0018/
       DATA CODE(1, 30,1),CODE(2, 30,1),CODE(3, 30,1)/ 8, 31,Z0002/
       DATA CODE(1, 31,1),CODE(2, 31,1),CODE(3, 31,1)/ 8, 32,Z0003/
       DATA CODE(1, 32,1),CODE(2, 32,1),CODE(3, 32,1)/ 8, 33,Z001A/
       DATA CODE(1, 33,1),CODE(2, 33,1),CODE(3, 33,1)/ 8, 34,Z001B/
       DATA CODE(1, 34,1),CODE(2, 34,1),CODE(3, 34,1)/ 8, 35,Z0012/
       DATA CODE(1, 35,1),CODE(2, 35,1),CODE(3, 35,1)/ 8, 36,Z0013/
       DATA CODE(1, 36,1),CODE(2, 36,1),CODE(3, 36,1)/ 8, 37,Z0014/
       DATA CODE(1, 37,1),CODE(2, 37,1),CODE(3, 37,1)/ 8, 38,Z0015/
       DATA CODE(1, 38,1),CODE(2, 38,1),CODE(3, 38,1)/ 8, 39,Z0016/
       DATA CODE(1, 39,1),CODE(2, 39,1),CODE(3, 39,1)/ 8, 40,Z0017/
       DATA CODE(1, 40,1),CODE(2, 40,1),CODE(3, 40,1)/ 8, 41,Z0028/
       DATA CODE(1, 41,1),CODE(2, 41,1),CODE(3, 41,1)/ 8, 42,Z0029/
       DATA CODE(1, 42,1),CODE(2, 42,1),CODE(3, 42,1)/ 8, 43,Z002A/
       DATA CODE(1, 43,1),CODE(2, 43,1),CODE(3, 43,1)/ 8, 44,Z002B/
       DATA CODE(1, 44,1),CODE(2, 44,1),CODE(3, 44,1)/ 8, 45,Z002C/
       DATA CODE(1, 45,1),CODE(2, 45,1),CODE(3, 45,1)/ 8, 46,Z002D/
       DATA CODE(1, 46,1),CODE(2, 46,1),CODE(3, 46,1)/ 8, 47,Z0004/
       DATA CODE(1, 47,1),CODE(2, 47,1),CODE(3, 47,1)/ 8, 48,Z0005/
       DATA CODE(1, 48,1),CODE(2, 48,1),CODE(3, 48,1)/ 8, 49,Z000A/
       DATA CODE(1, 49,1),CODE(2, 49,1),CODE(3, 49,1)/ 8, 50,Z000B/
       DATA CODE(1, 50,1),CODE(2, 50,1),CODE(3, 50,1)/ 8, 51,Z0052/
       DATA CODE(1, 51,1),CODE(2, 51,1),CODE(3, 51,1)/ 8, 52,Z0053/
       DATA CODE(1, 52,1),CODE(2, 52,1),CODE(3, 52,1)/ 8, 53,Z0054/
```

```
DATA CODE(1, 53,1),CODE(2, 53,1),CODE(3, 53,1)/ 8, 54,Z0055/
DATA CODE(1, 54,1),CODE(2, 54,1),CODE(3, 54,1)/ 8, 55,Z0024/
DATA CODE(1, 55,1),CODE(2, 55,1),CODE(3, 55,1)/ 8, 56,Z0025/
DATA CODE(1, 56,1),CODE(2, 56,1),CODE(3, 56,1)/ 8, 57,Z0058/
DATA CODE(1, 57,1),CODE(2, 57,1),CODE(3, 57,1)/ 8, 58,Z0059/
DATA CODE(1, 58,1),CODE(2, 58,1),CODE(3, 58,1)/ 8, 59,Z005A/
DATA CODE(1, 59,1),CODE(2, 59,1),CODE(3, 59,1)/ 8, 60,Z005B/
DATA CODE(1, 60,1),CODE(2, 60,1),CODE(3, 60,1)/ 8, 61,Z004A/
DATA CODE(1, 61,1),CODE(2, 61,1),CODE(3, 61,1)/ 8, 62,Z004B/
DATA CODE(1, 62,1),CODE(2, 62,1),CODE(3, 62,1)/ 8, 63,Z0032/
DATA CODE(1, 63,1),CODE(2, 63,1),CODE(3, 63,1)/ 8, 64,Z0033/
DATA CODE(1, 64,1),CODE(2, 64,1),CODE(3, 64,1)/ 8, 69,Z0034/
DATA CODE(1, 65,1),CODE(2, 65,1),CODE(3, 65,1)/ 5, 66,Z0018/
DATA CODE(1, 66,1),CODE(2, 66,1),CODE(3, 66,1)/ 5, 67,Z0012/
DATA CODE(1, 67,1),CODE(2, 67,1),CODE(3, 67,1)/ 6,  2,Z0017/
DATA CODE(1, 68,1),CODE(2, 68,1),CODE(3, 68,1)/ 7, 30,Z0037/
DATA CODE(1, 69,1),CODE(2, 69,1),CODE(3, 69,1)/ 8,  1,Z0036/
DATA CODE(1, 70,1),CODE(2, 70,1),CODE(3, 70,1)/ 8, 71,Z0037/
DATA CODE(1, 71,1),CODE(2, 71,1),CODE(3, 71,1)/ 8, 72,Z0064/
DATA CODE(1, 72,1),CODE(2, 72,1),CODE(3, 72,1)/ 8, 73,Z0065/
DATA CODE(1, 73,1),CODE(2, 73,1),CODE(3, 73,1)/ 8, 74,Z0068/
DATA CODE(1, 74,1),CODE(2, 74,1),CODE(3, 74,1)/ 8, 75,Z0067/
DATA CODE(1, 75,1),CODE(2, 75,1),CODE(3, 75,1)/ 9, 76,Z00CC/
DATA CODE(1, 76,1),CODE(2, 76,1),CODE(3, 76,1)/ 9, 77,Z00CD/
DATA CODE(1, 77,1),CODE(2, 77,1),CODE(3, 77,1)/ 9, 78,Z00D2/
DATA CODE(1, 78,1),CODE(2, 78,1),CODE(3, 78,1)/ 9, 79,Z00D3/
DATA CODE(1, 79,1),CODE(2, 79,1),CODE(3, 79,1)/ 9, 80,Z00D4/
DATA CODE(1, 80,1),CODE(2, 80,1),CODE(3, 80,1)/ 9, 81,Z00D5/
DATA CODE(1, 81,1),CODE(2, 81,1),CODE(3, 81,1)/ 9, 82,Z00D6/
DATA CODE(1, 82,1),CODE(2, 82,1),CODE(3, 82,1)/ 9, 83,Z00D7/
DATA CODE(1, 83,1),CODE(2, 83,1),CODE(3, 83,1)/ 9, 84,Z00D8/
DATA CODE(1, 84,1),CODE(2, 84,1),CODE(3, 84,1)/ 9, 85,Z00D9/
DATA CODE(1, 85,1),CODE(2, 85,1),CODE(3, 85,1)/ 9, 86,Z00DA/
DATA CODE(1, 86,1),CODE(2, 86,1),CODE(3, 86,1)/ 9, 87,Z00DB/
DATA CODE(1, 87,1),CODE(2, 87,1),CODE(3, 87,1)/ 9, 88,Z0098/
DATA CODE(1, 88,1),CODE(2, 88,1),CODE(3, 88,1)/ 9, 89,Z0099/
DATA CODE(1, 89,1),CODE(2, 89,1),CODE(3, 89,1)/ 9, 91,Z009A/
DATA CODE(1, 90,1),CODE(2, 90,1),CODE(3, 90,1)/ 6, 13,Z0018/
DATA CODE(1, 91,1),CODE(2, 91,1),CODE(3, 91,1)/ 9, 93,Z009B/
DATA CODE(1, 92,1),CODE(2, 92,1),CODE(3, 92,1)/12, 93,Z0001/
DATA CODE(1,  1,2),CODE(2,  1,2),CODE(3,  1,2)/10, 65,Z0037/
DATA CODE(1,  2,2),CODE(2,  2,2),CODE(3,  2,2)/ 3,  6,Z0002/
DATA CODE(1,  3,2),CODE(2,  3,2),CODE(3,  3,2)/ 2,  4,Z0003/
DATA CODE(1,  4,2),CODE(2,  4,2),CODE(3,  4,2)/ 2,  5,Z0002/
DATA CODE(1,  5,2),CODE(2,  5,2),CODE(3,  5,2)/ 3,  2,Z0003/
DATA CODE(1,  6,2),CODE(2,  6,2),CODE(3,  6,2)/ 4,  7,Z0003/
DATA CODE(1,  7,2),CODE(2,  7,2),CODE(3,  7,2)/ 4,  8,Z0002/
DATA CODE(1,  8,2),CODE(2,  8,2),CODE(3,  8,2)/ 5,  9,Z0003/
DATA CODE(1,  9,2),CODE(2,  9,2),CODE(3,  9,2)/ 6, 10,Z0005/
DATA CODE(1, 10,2),CODE(2, 10,2),CODE(3, 10,2)/ 6, 11,Z0004/
DATA CODE(1, 11,2),CODE(2, 11,2),CODE(3, 11,2)/ 7, 12,Z0004/
DATA CODE(1, 12,2),CODE(2, 12,2),CODE(3, 12,2)/ 7, 13,Z0005/
DATA CODE(1, 13,2),CODE(2, 13,2),CODE(3, 13,2)/ 7, 14,Z0007/
DATA CODE(1, 14,2),CODE(2, 14,2),CODE(3, 14,2)/ 8, 15,Z0004/
DATA CODE(1, 15,2),CODE(2, 15,2),CODE(3, 15,2)/ 8, 16,Z0007/
DATA CODE(1, 16,2),CODE(2, 16,2),CODE(3, 16,2)/ 9, 17,Z0018/
DATA CODE(1, 17,2),CODE(2, 17,2),CODE(3, 17,2)/10, 18,Z0017/
DATA CODE(1, 18,2),CODE(2, 18,2),CODE(3, 18,2)/10, 19,Z0018/
DATA CODE(1, 19,2),CODE(2, 19,2),CODE(3, 19,2)/10,  1,Z0008/
DATA CODE(1, 20,2),CODE(2, 20,2),CODE(3, 20,2)/11, 21,Z0067/
DATA CODE(1, 21,2),CODE(2, 21,2),CODE(3, 21,2)/11, 22,Z0068/
DATA CODE(1, 22,2),CODE(2, 22,2),CODE(3, 22,2)/11, 23,Z006C/
DATA CODE(1, 23,2),CODE(2, 23,2),CODE(3, 23,2)/11, 24,Z0037/
DATA CODE(1, 24,2),CODE(2, 24,2),CODE(3, 24,2)/11, 25,Z0028/
DATA CODE(1, 25,2),CODE(2, 25,2),CODE(3, 25,2)/11, 26,Z0017/
DATA CODE(1, 26,2),CODE(2, 26,2),CODE(3, 26,2)/11, 27,Z0018/
DATA CODE(1, 27,2),CODE(2, 27,2),CODE(3, 27,2)/12, 28,Z00CA/
DATA CODE(1, 28,2),CODE(2, 28,2),CODE(3, 28,2)/12, 29,Z00CB/
DATA CODE(1, 29,2),CODE(2, 29,2),CODE(3, 29,2)/12, 30,Z00CC/
DATA CODE(1, 30,2),CODE(2, 30,2),CODE(3, 30,2)/12, 31,Z00CD/
DATA CODE(1, 31,2),CODE(2, 31,2),CODE(3, 31,2)/12, 32,Z0068/
DATA CODE(1, 32,2),CODE(2, 32,2),CODE(3, 32,2)/12, 33,Z0069/
DATA CODE(1, 33,2),CODE(2, 33,2),CODE(3, 33,2)/12, 34,Z006A/
DATA CODE(1, 34,2),CODE(2, 34,2),CODE(3, 34,2)/12, 35,Z006B/
DATA CODE(1, 35,2),CODE(2, 35,2),CODE(3, 35,2)/12, 36,Z00D2/
DATA CODE(1, 36,2),CODE(2, 36,2),CODE(3, 36,2)/12, 37,Z00D3/
DATA CODE(1, 37,2),CODE(2, 37,2),CODE(3, 37,2)/12, 38,Z00D4/
DATA CODE(1, 38,2),CODE(2, 38,2),CODE(3, 38,2)/12, 39,Z00D5/
DATA CODE(1, 39,2),CODE(2, 39,2),CODE(3, 39,2)/12, 40,Z00D6/
DATA CODE(1, 40,2),CODE(2, 40,2),CODE(3, 40,2)/12, 41,Z00D7/
DATA CODE(1, 41,2),CODE(2, 41,2),CODE(3, 41,2)/12, 42,Z006C/
DATA CODE(1, 42,2),CODE(2, 42,2),CODE(3, 42,2)/12, 43,Z006D/
```

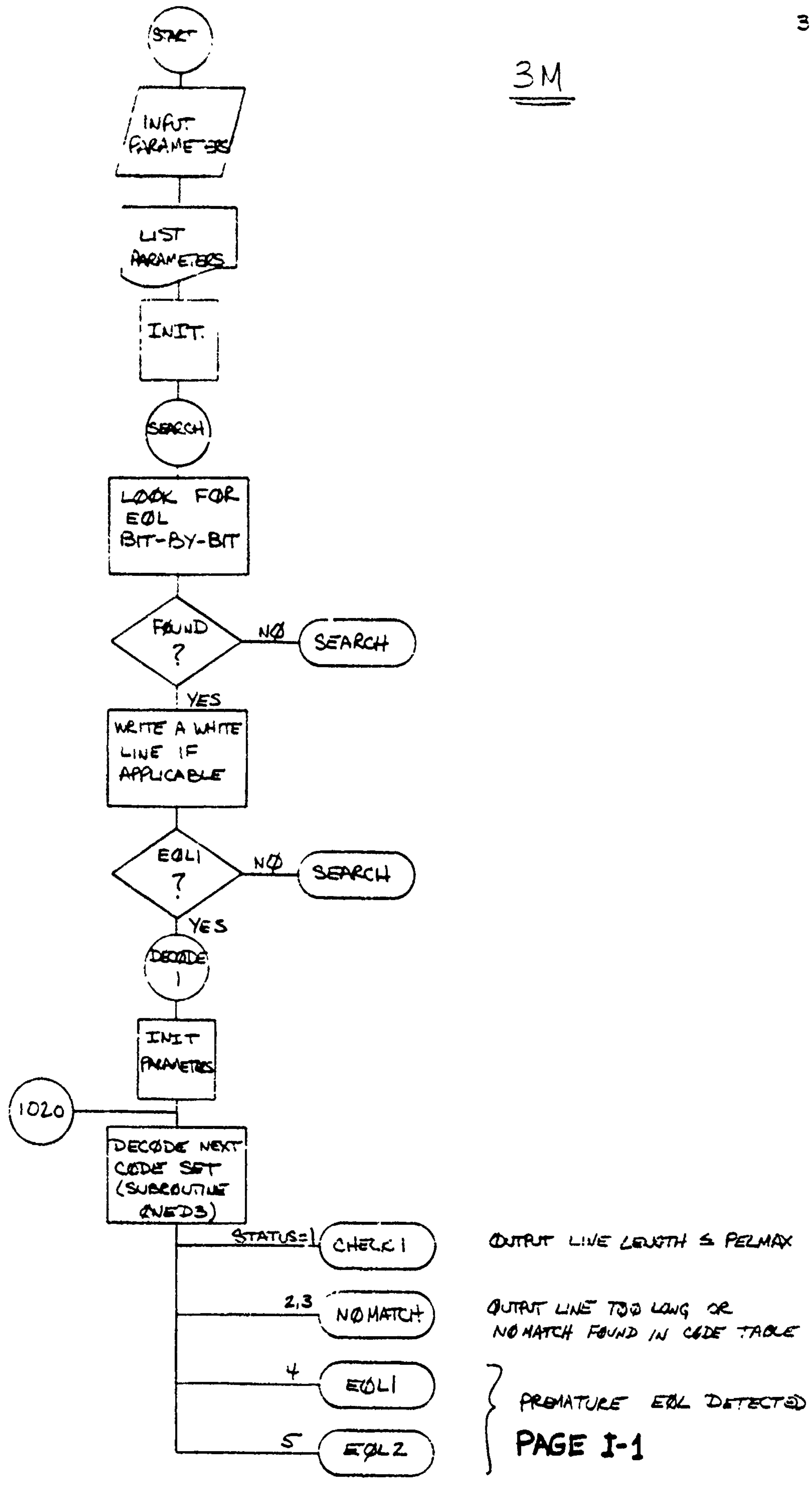
```
      DATA CODE(1, 43,2),CODE(2, 43,2),CODE(3, 43,2)/12, 44,Z000A/
      DATA CODE(1, 44,2),CODE(2, 44,2),CODE(3, 44,2)/12, 45,Z000B/
      DATA CODE(1, 45,2),CODE(2, 45,2),CODE(3, 45,2)/12, 46,Z0054/
      DATA CODE(1, 46,2),CODE(2, 46,2),CODE(3, 46,2)/12, 47,Z0055/
      DATA CODE(1, 47,2),CODE(2, 47,2),CODE(3, 47,2)/12, 48,Z0056/
      DATA CODE(1, 48,2),CODE(2, 48,2),CODE(3, 48,2)/12, 49,Z0057/
      DATA CODE(1, 49,2),CODE(2, 49,2),CODE(3, 49,2)/12, 50,Z0064/
      DATA CODE(1, 50,2),CODE(2, 50,2),CODE(3, 50,2)/12, 51,Z0065/
      DATA CODE(1, 51,2),CODE(2, 51,2),CODE(3, 51,2)/12, 52,Z0052/
      DATA CODE(1, 52,2),CODE(2, 52,2),CODE(3, 52,2)/12, 53,Z0053/
      DATA CODE(1, 53,2),CODE(2, 53,2),CODE(3, 53,2)/12, 54,Z0024/
      DATA CODE(1, 54,2),CODE(2, 54,2),CODE(3, 54,2)/12, 55,Z0037/
      DATA CODE(1, 55,2),CODE(2, 55,2),CODE(3, 55,2)/12, 56,Z0038/
      DATA CODE(1, 56,2),CODE(2, 56,2),CODE(3, 56,2)/12, 57,Z0027/
      DATA CODE(1, 57,2),CODE(2, 57,2),CODE(3, 57,2)/12, 58,Z0028/
      DATA CODE(1, 58,2),CODE(2, 58,2),CODE(3, 58,2)/12, 59,Z0058/
      DATA CODE(1, 59,2),CODE(2, 59,2),CODE(3, 59,2)/12, 60,Z0059/
      DATA CODE(1, 60,2),CODE(2, 60,2),CODE(3, 60,2)/12, 61,Z002B/
      DATA CODE(1, 61,2),CODE(2, 61,2),CODE(3, 61,2)/12, 62,Z002C/
      DATA CODE(1, 62,2),CODE(2, 62,2),CODE(3, 62,2)/12, 63,Z005A/
      DATA CODE(1, 63,2),CODE(2, 63,2),CODE(3, 63,2)/12, 64,Z0066/
      DATA CODE(1, 64,2),CODE(2, 64,2),CODE(3, 64,2)/12, 66,Z0067/
      DATA CODE(1, 65,2),CODE(2, 65,2),CODE(3, 65,2)/10, 20,Z000F/
      DATA CODE(1, 66,2),CODE(2, 66,2),CODE(3, 66,2)/12, 67,Z00C8/
      DATA CODE(1, 67,2),CODE(2, 67,2),CODE(3, 67,2)/12, 68,Z00C9/
      DATA CODE(1, 68,2),CODE(2, 68,2),CODE(3, 68,2)/12, 69,Z005B/
      DATA CODE(1, 69,2),CODE(2, 69,2),CODE(3, 69,2)/12, 70,Z0033/
      DATA CODE(1, 70,2),CODE(2, 70,2),CODE(3, 70,2)/12, 71,Z0034/
      DATA CODE(1, 71,2),CODE(2, 71,2),CODE(3, 71,2)/12, 72,Z0035/
      DATA CODE(1, 72,2),CODE(2, 72,2),CODE(3, 72,2)/13, 73,Z006C/
      DATA CODE(1, 73,2),CODE(2, 73,2),CODE(3, 73,2)/13, 74,Z006D/
      DATA CODE(1, 74,2),CODE(2, 74,2),CODE(3, 74,2)/13, 75,Z004A/
      DATA CODE(1, 75,2),CODE(2, 75,2),CODE(3, 75,2)/13, 76,Z004B/
      DATA CODE(1, 76,2),CODE(2, 76,2),CODE(3, 76,2)/13, 77,Z004C/
      DATA CODE(1, 77,2),CODE(2, 77,2),CODE(3, 77,2)/13, 78,Z004D/
      DATA CODE(1, 78,2),CODE(2, 78,2),CODE(3, 78,2)/13, 79,Z0072/
      DATA CODE(1, 79,2),CODE(2, 79,2),CODE(3, 79,2)/13, 80,Z0073/
      DATA CODE(1, 80,2),CODE(2, 80,2),CODE(3, 80,2)/13, 81,Z0074/
      DATA CODE(1, 81,2),CODE(2, 81,2),CODE(3, 81,2)/13, 82,Z0075/
      DATA CODE(1, 82,2),CODE(2, 82,2),CODE(3, 82,2)/13, 83,Z0076/
      DATA CODE(1, 83,2),CODE(2, 83,2),CODE(3, 83,2)/13, 84,Z0077/
      DATA CODE(1, 84,2),CODE(2, 84,2),CODE(3, 84,2)/13, 85,Z0052/
      DATA CODE(1, 85,2),CODE(2, 85,2),CODE(3, 85,2)/13, 86,Z0053/
      DATA CODE(1, 86,2),CODE(2, 86,2),CODE(3, 86,2)/13, 87,Z0054/
      DATA CODE(1, 87,2),CODE(2, 87,2),CODE(3, 87,2)/13, 88,Z0055/
      DATA CODE(1, 88,2),CODE(2, 88,2),CODE(3, 88,2)/13, 89,Z005A/
      DATA CODE(1, 89,2),CODE(2, 89,2),CODE(3, 89,2)/13, 90,Z005B/
      DATA CODE(1, 90,2),CODE(2, 90,2),CODE(3, 90,2)/13, 91,Z0064/
      DATA CODE(1, 91,2),CODE(2, 91,2),CODE(3, 91,2)/13, 93,Z0065/
      DATA CODE(1, 92,2),CODE(2, 92,2),CODE(3, 92,2)/12, 72,Z0001/
      DATA CODERD(1,1),CODERD(2,1),CODERD(3,1)/          4,  2,Z     E/
      DATA CODERD(1,2),CODERD(2,2),CODERD(3,2)/          4,  8,Z     F/
      DATA CODERD(1,3),CODERD(2,3),CODERD(3,3)/          1,  4,Z     0/
      DATA CODERD(1,4),CODERD(2,4),CODERD(3,4)/          3,  6,Z     4/
      DATA CODERD(1,5),CODERD(2,5),CODERD(3,5)/          4,  7,Z     C/
      DATA CODERD(1,6),CODERD(2,6),CODERD(3,6)/          3,  5,Z     5/
      DATA CODERD(1,7),CODERD(2,7),CODERD(3,7)/          4,  1,Z     D/
      DATA CODERD(1,8),CODERD(2,8),CODERD(3,8)/          8,  9,Z    7F/
      DATA CODERD(1,9),CODERD(2,9),CODERD(3,9)/          8, 10,Z    7E/
C
      E N D
0        END OF DCEC UPRINT PROGRAM          LINES PRINTED=  1536
```

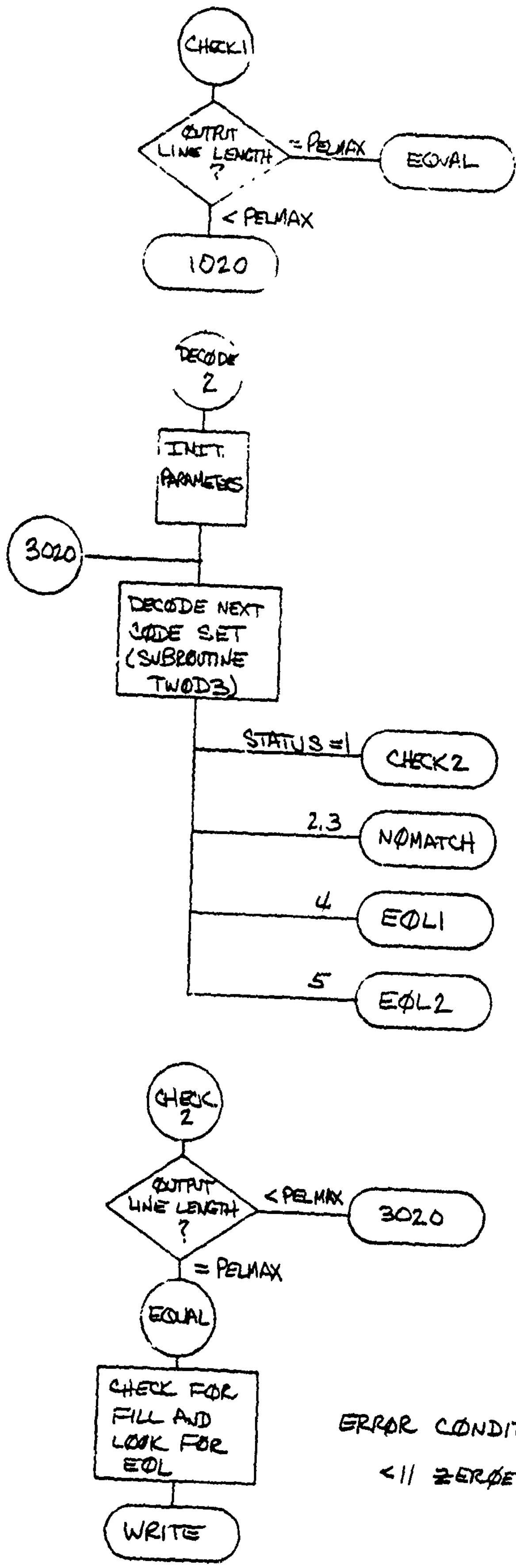
APPENDIX I

PROGRAM FLOW CHART

FOR 3M ALGORITHM

3M


START
INPUT PARAMETERS
LIST PARAMETERS
INIT.
SEARCH
LOOK FOR EOL BIT-BY-BIT
FOUND ?
NO
SEARCH
YES
WRITE A WHITE LINE IF APPLICABLE
EOL1 ?
NO
SEARCH
YES
DECODE 1
INIT PARAMETERS
1020
DECODE NEXT CODE SET (SUBROUTINE QWED3)
STATUS=1
CHECK1
OUTPUT LINE LENGTH ≤ PELMAX
2,3
NOMATCH
OUTPUT LINE TOO LONG OR NO MATCH FOUND IN CODE TABLE
4
EOL1
5
EOL2
PREMATURE EOL DETECTED



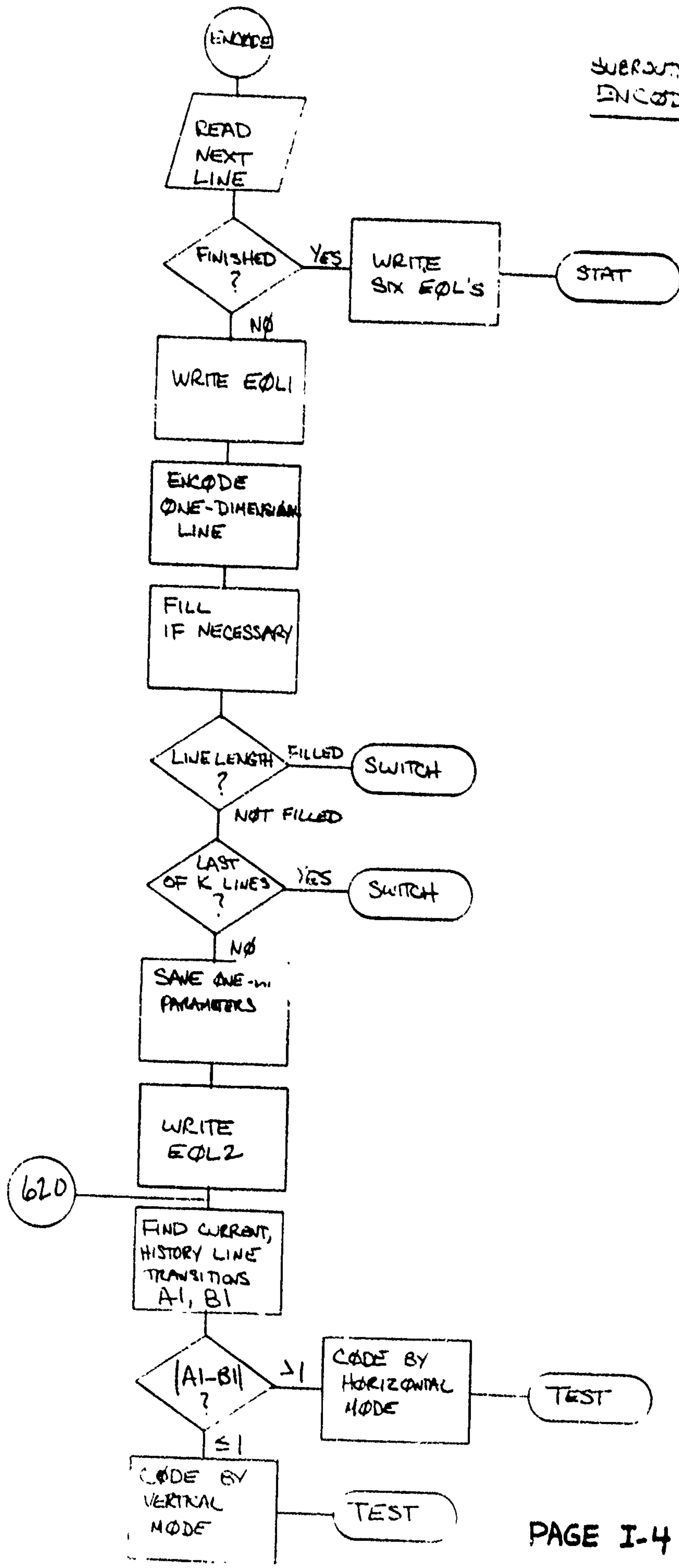
ERROR CONDITION DETECTED:

<11 ZEROES IN EOL

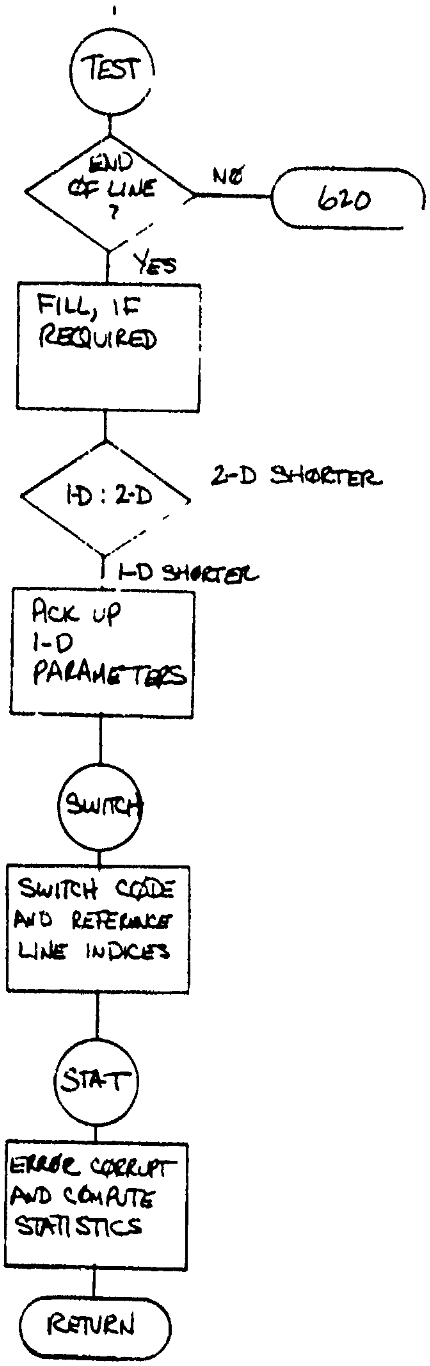
WRITE

REFER TO PAGE 3 OF IBM FLOWCHART

SUBROUTINE ENCOD3

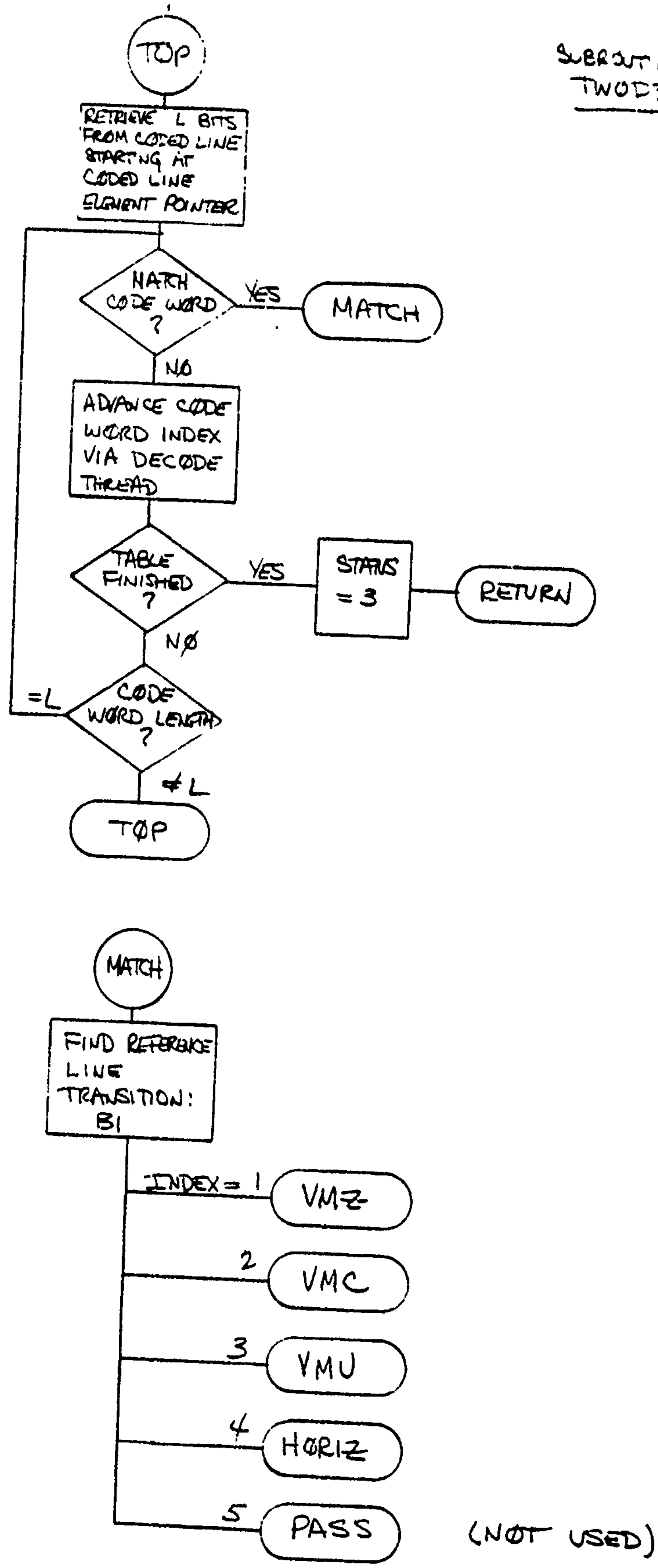


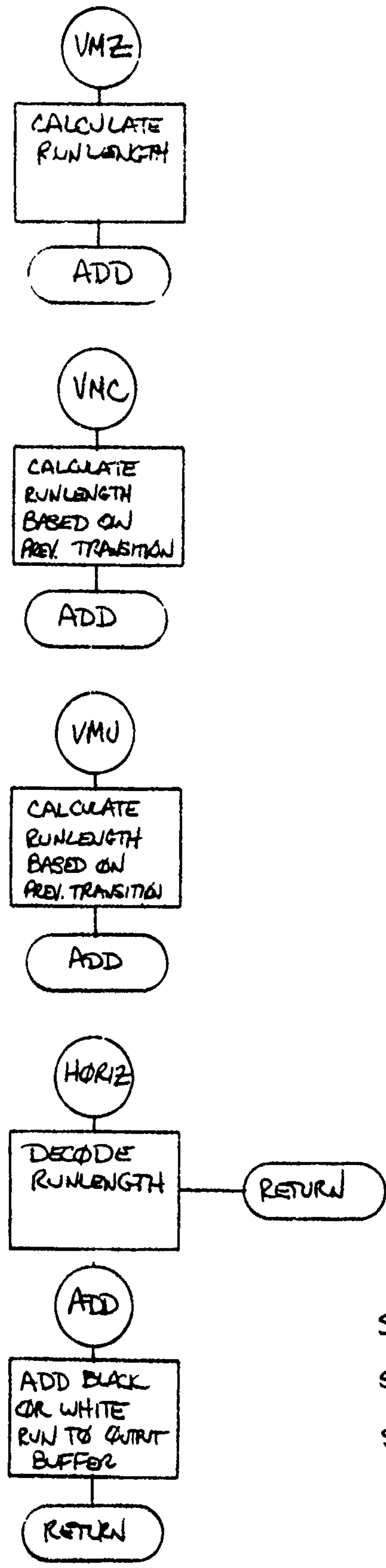
3-5/7


TEST
END OF LINE ?
NO
620
YES
FILL, IF REQUIRED
1-D : 2-D
2-D SHORTER
1-D SHORTER
PICK UP 1-D PARAMETERS
SWITCH
SWITCH CODE AND REFERENCE LINE INDICES
STAT
ERROR CORRUPT AND COMPUTE STATISTICS
RETURN

SUBROUTINE TWOD3

3-6/7 |





STATUS = 1: LINE LENGTH ≤ PELMAX

STATUS = 2: LINE TOO LONG (> PELMAX)

STATUS = 3: NEGATIVE RUNLENGTH

APPENDIX J

COMPUTER PROGRAM CODE LISTING

3M ALGORITHM

```
START OF DCEC JPRINT PROGRAM          DSNAME=D0031.THREEM.FORT
C       PROGRAM THREEM
        IMPLICIT INTEGER(A-Z)
        REAL CF3,CF4,ERRATE
C******* LABELED COMMON /G32BIT/ *******
C
        COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
        INTEGER MASK,COMASK,LIBIT,LZBIT
C
        COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
       *            STFBUF(240), STAT(3000)
        COMMON/HUFF/CODE(3,92,2),CCDERD(3,9)
        COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
        COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C******************** LABELLED COMMON VARIABLES **********************
C
        COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
        COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
       *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
       *            ERRCNT,INLNCT,CONSEC,LNNOBF,KCNT,
       *            INCOD,INREF,OTCOD,OTREF,STFBIT,VMMD
        COMMON/ICHAR/DD,II,MM,TT,NN,YY
        COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
        LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C    READ INPUT PARAMETERS
   90 WRITE(TERM,100)
  100 FORMAT('$PARAMETERS: INPUT(=I), OR DEFAULT(=D)?')
      READ(TERM,110,ERR=90) INSW
  110 FORMAT(A1)
      IF (INSW.EQ.DD) GO TO 315
      IF (INSW.NE.II) GO TO 90
C
C    READ DIAGNOSTIC SWITCH
C
  114 WRITE(TERM,115)
  115 FORMAT('$DIAGNOSTIC PRINTOUT? (Y OR N): ')
      READ(TERM,110) INSW
      IF(INSW.EQ.YY) GO TO 116
      IF(INSW.EQ.NN) GO TO 120
      GO TO 114
  116 CONTINUE
      DIAG=.TRUE.
C
C    READ MAXIMUM NUMBER OF PELS PER LINE
C
  120 CONTINUE
      WRITE(TERM,130)
  130 FORMAT('$ENTER MAXIMUM NUMBER OF PELS PER LINE: ')
      READ(TERM,140,ERR=120) PELMAX
  140 FORMAT(I4)
      IF(PELMAX.GE.1.AND.PELMAX.LE.1728) GO TO 160
      WRITE(TERM,150) PELMAX
  150 FORMAT('ONUMBER OUT OF RANGE (=',I6,')')
      GO TO 120
C
C    READ VERTICAL SAMPLING
C
  160 CONTINUE
      WRITE(TERM,170)
  170 FORMAT('$ENTER VERTICAL SAMPLING: ')
      READ(TERM,180,ERR=160) VRES
  180 FORMAT(I2)
      IF(VRES.GE.1.AND.VRES.LE.10) GO TO 190
      WRITE(TERM,150) VRES
      GO TO 160
C
C    READ PARAMETER K
C
  190 CONTINUE
      WRITE(TERM,192)
  192 FORMAT('$ENTER PARAMETER K: ')
      READ(TERM,140,ERR=190) K
      IF(K.GE.1.AND.K.LE.3000) GO TO 200
      WRITE(TERM,150) K
      GO TO 190
C
C    READ ERROR PATTERN PHASE
C
```

```
  200 CONTINUE
      WRITE(TERM,210)
  210 FORMAT('$ENTER ERROR PATTERN PHASE: ')
      READ(TERM,220,ERR=200) EPHASE
  220 FORMAT(I1)
      IF(EPHASE.GE.0.AND.EPHASE.LE.3) GO TO 240
      WRITE(TERM,150) EPHASE
      GO TO 200
C
C   READ MINIMUM COMPRESSED LINE LENGTH
C
  240 CONTINUE
      WRITE(TERM,250)
  250 FORMAT('$ENTER MINIMUM COMPRESSED LINE LENGTH: ')
      READ(TERM,140,ERR=240) CMPMAX
      IF(CMPMAX.GE.0.AND.CMPMAX.LE.1728) GO TO 320
      WRITE(TERM,150) CMPMAX
      GO TO 240
C
C   READ NUMBER OF SCAN LINES TO BE PROCESSED
  320 CONTINUE
      WRITE(TERM,330)
  330 FORMAT('$NUMBER OF SCAN LINES TO BE PROCESSED=? ')
      READ(TERM,140,ERR=320) LINMAX
      IF(LINMAX.GE.1.AND.LINMAX.LE.3000) GO TO 280
      WRITE(TERM,150) LINMAX
      GO TO 320
C
C   READ ERROR MODE
C
  280 CONTINUE
      WRITE(TERM,290)
  290 FORMAT('$ERROR MODE=? (M=MANUAL,T=TAPE,N=NO ERRORS)')
      READ(TERM,110,ERR=280) ERRMOD
      IF(ERRMOD.EQ.MM) GO TO 300
      IF(ERRMOD.EQ.TT) GO TO 315
      IF(ERRMOD.NE.NN) GO TO 280
      GO TO 350
C
C   READ ERROR LOCATIONS
C
  300 CONTINUE
      ERRLIM=1
  305 READ(TERM,140) ERRORS(ERRLIM)
      IF(ERRORS(ERRLIM).EQ.9999) GO TO 310
      ERRLIM=ERRLIM+1
      GO TO 30
  310 CONTINUE
      ERRLIM=ERRLIM-1
      GO TO 350
C
C   READ ERROR TAPE FILE AND OPEN
C
  315 CONTINUE
C
      ERRLIM=1
      READ(ERFIL,318,END=317) ERRORS(ERRLIM)
      ERRLIM=ERRLIM+1
  316 READ(ERFIL,318,END=317) ERRORS(ERRLIM)
  318 FORMAT(I16)
      ERRORS(ERRLIM)=ERRORS(ERRLIM)+ERRORS(ERRLIM-1)
      ERRLIM=ERRLIM+1
      GO TO 316
  317 ERRLIM=ERRLIM-1
C
  350 CONTINUE
C
  360 CONTINUE
C   WRITE INPUT PARAMETERS
C
      WRITE(LPFIL,400) PELMAX,VRES,K,EPHASE,CMPMAX,LINMAX
  400 FORMAT('1INPUT PARAMETERS:'/
     *       '0MAXIMUM NUMBER OF PELS PER LINE=',I6/
     *       '0VERTICAL SAMPLING: N=',I4/
     *       '0PARAMETER K =',I4/
     *       '0ERROR PATTERN PHASE =',I4/
     *       '0MINIMUM COMPRESSED LINE LENGTH =',I4,' BITS'/
     *       '0NUMBER OF SCAN LINES TO BE PROCESSED =',I6)
      IF(ERRMOD.EQ.NN) WRITE(LPFIL,410)
  410 FORMAT ('0NO ERRORS INSERTED')
      IF(ERRMOD.EQ.MM) WRITE(TERM,140) (ERRORS(I),I=1,ERRLIM)
      IF(ERRMOD.EQ.TT) WRITE(TERM,420) ERRLIM
```

```
  420 FORMAT(I12,' ERRORS OBTAINED FROM ERROR TAPE')
C********************* BEGIN PROGRAM *************************************
C
C    INITIALIZE
C
      TCDEL=0
      TCDATA=0
      ERRPNT=1
      ERRCNT=0
      INLNCT=0
      ERROFF=EPHASE*1024
      CDELCT=32
      OTELP=1
      CDELP=32+1
      CONSEC=1
      INREF=1
      INCOD=2
      OTREF=1
      OTCOD=2
      KCNT=K
      STFBIT=0
C
      DO 800 I=1,240
      STFBUF(I)=0
      CDBUF(I)=0
  800 CONTINUE
      DO 850 I=1,60
      OTBUF(I,OTREF)=0
      OTBUF(I,OTCOD)=0
      PELBUF(I,INREF)=0
      PELBUF(I,INCOD)=0
  850 CONTINUE
      SEARCH=.TRUE.
      SYNC=.FALSE.
      WRITE=.FALSE.
C
C    SEARCH MODE: LOOK FOR EOL1 BIT-BY-BIT
C
 900  CONTINUE
      CALL GETL3(13,MODE,LBITS,L)
      GO TO (910,930,930,920),MODE
      STOP 900
 910  CONTINUE
C
C    EOL NOT FOUND; ADVANCE POINTER AND TRY AGAIN
C
      CDELP=CDELP+1
      GO TO 900
 920  CONTINUE
      STOP 920
 930  CONTINUE
C
C    EOL FOUND
C
      SEARCH=.FALSE.
      CDELP=CDELP+L
      IF(WRITE) GO TO 935
      WRITE=.TRUE.
      GO TO 960
 935  CONTINUE
C
C    SET OUTPUT DECODE LINE TO 0 AND WRITE OUT
      DO 950 I=1,60
      OTBUF(I,OTCOD)=0
 950  CONTINUE
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
 960  CONTINUE
      IF(MODE-2)965,1000,900
 965  STOP 965
 1000 CONTINUE
C
C    PERFORM ONE-DIMENSIONAL DECODE OF A COMPLETE LINE
C    FIRST,SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 1010 I=1,60
      OTBUF(I,OTCOD)=0
 1010 CONTINUE
C
      INDEX=3
      COLOR=1
```

```
      OTELP=1
C
 1020 CONTINUE
      CALL ONED3(INDEX,COLOR,STATUS,L)
      GO TO (1030,1070,1070,1035,1040),STATUS
C            1    2    3    4    5
      STOP 1000
C
C     RUN ADDED; CHECK LENGTH OF OUTPUT LINE
C
 1030 CONTINUE
      ONE=.TRUE.
      IF(OTELP-1-PELMAX) 1031,1032,1050
 1031 CONTINUE
      IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
      INDEX=3
      GO TO 1020
 3000 CONTINUE
C
C     PERFORM TWO-DIMENSIONAL DECODE
C
C
C     FIRST,SET OUTPUT BUFFER TO WHITE
C     (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 3010 I=1,60
      OTBUF(I,OTCOD)=0
 3010 CONTINUE
C
      INDEX=1
      COLOR=1
      OTELP=1
      VMMD=0
C
 3020 CONTINUE
      CALL TWOD3(INDEX,COLOR,STATUS,L)
      GO TO (3030,1070,1070,1035,1040),STATUS
C            1    2    3    4    5
      STOP 3000
C
C     RUN ADDED; LOOK FOR NEXT RUN
C
 3030 CONTINUE
      ONE=.FALSE.
      IF(OTELP-1-PELMAX) 3031,1032,1050
 3031 CONTINUE
      IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
      INDEX=1
      GO TO 3020
C
C
C     LINE LENGTH=PELMAX; CHECK FOR FILL AND LOOK FOR EOL
C
 1032 CONTINUE
      ZERO=-1
 1033 CONTINUE
      ZERO=ZERO+1
      CALL GETL3(1,MODE,LBITS,L)
C
      GO TO (1034,1050,1050,1050),MODE
C
C     CHECK FOR FILL
C
 1034 CONTINUE
C
      CDELP=CDELP+L
      IF(LBITS.EQ.0) GO TO 1033
C
C     ONE DETECTED; CHECK NUMBER OF CONSECUTIVE ZEROES
C
      IF(ZERO.LE.10) GO TO 1070
C
C     EOL FOUND; CHECK TYPE
C
      CALL GETL3(1,MODE,LBITS,L)
      IF(LBITS.EQ.0) MODE=2
      IF(LBITS.EQ.1) MODE=3
      GO TO (1070,1060,1060,1080),MODE
C
C     PREMATURE EOL DETECTED
C
C
```

```
C     EOL1 DETECTED
C
 1035 CONTINUE
      CDELP=CDELP+L
      STATUS=4
      IF(OTELP.LE.5) CONSEC=CONSEC+1
      IF(CONSEC-2)1080,1000,2000
C
C     EOL2 DETECTED
C
 1040 CONTINUE
      CDELP=CDELP+L
      STATUS=5
C
      GO TO 1080
C
C     PROBLEMS,PROBLEMS
C
 1050 STOP 1050
C
C     LINE LENGTH CORRECT, EOL DETECTED PROPERLY; WRITE OUTPUT LINE
C
 1060 CONTINUE
      CDELP=CDELP+L
      WRITE(OTFIL)OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
      CONSEC=1
      IF(ONE) SYNC=.TRUE.
      TEMP=OTREF
      OTREF=OTCOD
      OTCOD=TEMP
      IF(MODE.EQ.2) GO TO 1000
      GO TO 3000
C
C     LINE TOO LONG OR NO MATCH
C
 1070 CONTINUE
      WRITE=.FALSE.
C
C     LINE SHORT
C
 1080 CONTINUE
      IF(.NOT.SYNC) GO TO 1090
C
C     WRITE LAST GOOD LINE
C
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTREF),I=1,60)
      SYNC=.FALSE.
      GO TO 1110
 1090 CONTINUE
C
C     WRITE A WHITE LINE
C
      DO 1100 I=1, 60
 1100 OTBUF(I,OTCOD)=0
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
 1110 OTLNNO=LNNOBF
      IF(STATUS.EQ.4) GO TO 1000
      SEARCH=.TRUE.
      GO TO 900
C
C     END OF MESSAGE
C
 2000 CONTINUE
      WRITE(LPFIL,2010) CONSEC
 2010 FORMAT('0END OF MESSAGE DETECTED (',I2,' EOL''S)')
C
C     REPORT COMPRESSION FACTOR, ERROR SENSITIVITY FACTOR,BIT ERROR RATE
C
      ERRATE=FLOAT(ERRCNT)/FLOAT(TCDEL)
      WRITE(LPFIL,2020) TCDEL,TCDATA,STFBIT,INLNCT,ERRATE
 2020 FORMAT('0TOTAL NUMBER OF CODED BITS = ',I8/
     *       '0TOTAL NUMBER OF CODED DATA BITS = ',I8/
     *       '0TOTAL NUMBER OF 2-DIM LINES = ',I8/
     *       '0TOTAL NUMBER OF INPUT LINES PROCESSED = ',I8/
     *       '0BIT ERROR RATE = ',G14.6)
C
      CALL STATS(STAT,INLNCT,DIAG)
      CF3=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDEL)
      CF4=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDATA)
C
      WRITE(LPFIL,2030) CF3,CF4
```

```
 2030 FORMAT('0COMPRESSION FACTOR FOR G3 MACHINE (CF3) =',F8.4/
     *        '0COMPRESSION FACTOR FOR G4 MACHINE (CF4) =',F8.4)
C
      CALL ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      STOP
      END
      SUBROUTINE GETL3(LBITS,MODE,WRD,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *           CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *           ERRCNT,INLNCT,CONSEC,LNNOBF,KCNT,
     *           INCOD,INREF,OTCOD,OTREF,STFBIT,VMMD
      COMMON/ICHAR/OD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C**************************** BEGIN PROGRAM ***************************
C
      MODE=4
C
C   RETRIEVE NEXT BIT FROM CDBUF
C
 100  CONTINUE
C
C   ENCODE A NEW LINE IF NECESSARY
C
      IF(LBITS+CDELP-1.LE.CDELCT) GO TO 200
      IF(CDELCT-CDELP+1) 170,190,180
  170 STOP 170
  180 CONTINUE
      STFBUF(1)=I48(STFBUF,CDELP,CDELCT-CDELP+1)
  190 CONTINUE
      CDELP=32-(CDELCT-CDELP)
      CALL ENCOD3
 200  CONTINUE
      WRD=I48(STFBUF,CDELP,LBITS)
      L=LBITS
      IF(L.LT.13) GO TO 250
      IF(WRD.EQ.CODERD(3,6)) GO TO 300
      IF(WRD.EQ.CODERD(3,7)) GO TO 400
  250 CONTINUE
      MODE=1
      RETURN
  300 CONTINUE
      MODE=2
      RETURN
  400 CONTINUE
      MODE=3
      RETURN
      END
      SUBROUTINE ENCOD3
C
      IMPLICIT INTEGER(A-Z)
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
```

```
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *             CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *             ERRCNT,INLNCT,CONSEC,LNNOBF,KCNT,
     *             INCOD,INREF,OTCOD,OTREF,STFBIT,VMMD
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C*********************************** BEGIN PROGRAM *******************
C
C   INITIALIZE VARIABLES
C
      CDELCT=32
      CDDATA=0
      DO 50 I=2,240
      CDBUF(I)=0
   50 CONTINUE
C
C   READ INPUT PICTURE FILE
C
  100 CONTINUE
      READ(PELFIL,END=120,ERR=500)
     * INLNNO,INELCT,(PELBUF(I,INCOD),I=1,60)
      IF(MOD(INLNNO,100).EQ.0) WRITE(TERM,110) INLNNO
  110 FORMAT(' INPUT LINE NO. =',I6)
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
      IF(INELCT.LT.PELMAX) CALL EXIT
      INLNCT=INLNCT+1
C
C   LOAD OUTPUT LINE NUMBER BUFFER
C
      LNNOBF=INLNNO
      IF(SEARCH)OTLNNO=LNNOBF
C
      IF(INLNNO.LE.LINMAX) GO TO 140
C
C   WRITE SIX EOL1'S
  120 CONTINUE
      DO 130 I=1,6
      CALL CODE3M(6,0,0,0,0,CDELCT,CDDATA)
  130 CONTINUE
      DO 135 I=1,6
      STFBUF(I)=CDBUF(I)
  135 CONTINUE
      GO TO 345
  140 CONTINUE
C
C   ONE-DIMENSIONAL CODING
C   WRITE ONE EOL1
C
      CALL CODE3M(6,0,0,0,0,CDELCT,CDDATA)
C
      POLAR=1
C
C   TEST COLOR OF FIRST ELEMENT
C
      IF(I4B(PELBUF(1,INCOD),1,1).EQ.0) GO TO 150
C
C   FIRST ELEMENT BLACK; ENCODE 0-LENGTH WHITE RUN
C
      CALL CODELN(0,1,CDELCT,CDDATA)
      POLAR=2
C
C   CALCULATE RUN LENGTH AND ENCODE
C
  150 CONTINUE
      RUN=0
      DO 200 I=1,PELMAX
      PEL=I4B(PELBUF(1,INCOD),I,1)+1
      IF(PEL.EQ.POLAR) GO TO 180
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 170
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
  160 FORMAT(4I8)
  170 CONTINUE
      RUN=1
      POLAR=MOD(POLAR+2,2)+1
      GO TO 200
  180 CONTINUE
      RUN=RUN+1
  200 CONTINUE
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
```

```
      IF(.NOT.DIAG) GO TO 550
      WRITE(TERM,160) RJN,POLAR,CDELCT,CDDATA
  550 CONTINUE
C
C   FILL IF NECESSARY
C
      FILL=C[illegible]MAX-(CDELCT-32)
      IF(FILL) 559,559,555
  555 CONTINUE
      CDELCT=CDELCT+FILL
  559 CONTINUE
      CDELW=(CDELCT+32-1)/32
      DO 590 I=2,CDELW
      STFBUF(I)=CDBUF(I)
  590 CONTINUE
C
C   LINE FILLED; USE ONE-DIM
C
      IF(FILL) 560,560,320
  560 CONTINUE
C
C   TEST NUMBER OF CONSECUTIVE TWO-DIM LINES
C
      IF(KCNT-K) 580,320,570
  570 STOP 570
  580 CONTINUE
C
C   SAVE ONE-DIM PARAMETERS
C
      TEMPEL=CDELCT
      TEMPCD=CDDATA
C
C   TWO-DIMENSIONAL CODING
C
  600 CONTINUE
C
C   RE-INITIALIZE PARAMETERS
C
      CDELCT=32
      CDDATA=0
      DO 610 I=2,240
      CDBUF(I)=0
  610 CONTINUE
C
C   WRITE ONE EOL2
C
      CALL CODE3M(7,0,0,0,0,CDELCT,CDDATA)
C
C   SET A0 TO LEFT EDGE-1 AND POLARITY=WHITE
C
      A0=0
      POL=0
      LEFT=.TRUE.
      VMM=0
C
C   DETECT A1
C
  620 CONTINUE
      I=A0+1
      IF(I.GT.PELMAX) GO TO 640
  630 CONTINUE
      PEL=I48(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 640
      I=I+1
      IF(I.LE.PELMAX) GO TO 630
  640 CONTINUE
      A1=I
C
C   DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 665
      PELM1=I48(PELBUF(1,INREF),A0,1)
      IF(LEFT) PELM1=0
  650 CONTINUE
      PEL=I48(PELBUF(1,INREF),I,1)
      IF(PEL.NE.PELM1) GO TO 670
  660 CONTINUE
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 650
  665 CONTINUE
```

```
      B1=I
      GO TO 730
  670 CONTINUE
      IF(PEL.NE.POL) GO TO 690
      GO TO 660
  690 CONTINUE
      B1=I
C
C   ADJUST LEFT EDGE IF NECESSARY
C
  730 CONTINUE
      IF(.NOT.LEFT) POLAR=I48(PELBUF(1,INCCD),A0,1)+1
      IF(.NOT.LEFT) GO TO 740
      POLAR=1
      A0=1
      LEFT=.FALSE.
  740 CONTINUE
C
C   TEST FOR PASS MODE: OPTION NOT WELL DEFINED-SKIP
C
C     IF(B2.GE.A1) GO TO 750
C
C   PASS MODE CODING (CAN'T END A LINE IN PASS MODE;NEW A0 MUST HAVE
C                       SAME POLARITY AS B2)
C
C     CALL CODE3M(1,0,0,0,0,CDELCT,CDDATA)
C     A0=B2
C     GO TO 620
C 750 CONTINUE
C
C   CALCULATE VERTICAL DISTANCE
C
      MAB=IABS(A1-B1)
      IF(MAB-1) 840,850,790
C
  790 CONTINUE
C
C   CODE BY HORIZONTAL MODE;FIRST DETECT A2
C
      I=A1+1
      IF(I.GT.PELMAX) GO TO 810
C
C   CALCULATE POLARITY OF A1
C
      POL=I48(PELBUF(1,INCOD),A1,1)
  800 CONTINUE
      PEL=I48(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 820
      I=I+1
      IF(I.LE.PELMAX) GO TO 800
  810 A2=PELMAX+1
      GO TO 830
  820 CONTINUE
      A2=I
  830 CONTINUE
      CALL CODE3M(4,POLAR,A0,A1,A2,CDELCT,CDDATA)
      A0=A2
      GO TO 960
C
C   CODE BY VERTICAL MODE
C
C
  840 CALL CODE3M(1,0,0,0,0,CDELCT,CDDATA)
      GO TO 950
  850 IF(A1-B1) 870,860,880
C
  860 STOP 860
  870 CONTINUE
      IF(VMM) 871,872,873
  871 STOP 871
  872 CALL CODE3M(3,0,0,0,0,CDELCT,CDDATA)
      VMM=1
      GO TO 950
  873 CALL CODE3M(2,0,0,0,0,CDELCT,CDDATA)
      VMM=1
      GO TO 950
  880 CONTINUE
      IF(VMM) 881,882,883
  881 STOP 881
C
  882 CALL CODE3M(2,0,0,0,0,CDELCT,CDDATA)
      VMM=0
```

```
      GO TO 950
  883 CALL CODE3M(3,0,0,0,0,CDELCT,CDDATA)
      VMM=0
  950 CONTINUE
      A0=A1
C
C    TEST FOR END OF LINE
C
  960 CONTINUE
      IF(A0.GT.PELMAX) GO TO 210
      POL=I4B(PELBUF(1,INCCD),A0,1)
      GO TO 620
  210 CONTINUE
C
C    SAVE LINE LENGTH(DATA BITS ONLY)
C
      STAT(INLNCT)=CDDATA+13
C
C    CHECK CODED LINE LENGTH
C
      FILL=CMPMAX-(CDELCT-32)
      IF(FILL) 400,400,250
C
C    CODE LINE TOO SHORT; FILL IT TO CMPMAX
  250 CONTINUE
      CDELCT=CDELCT+FILL
  400 CONTINUE
C
C    COMPARE 1-D & 2-D CODED LENGTHS
C
      IF(TEMPEL.LE.CDELCT) GO TO 310
C
C    2-D SHORTER
C
      CDELW=(CDELCT+32-1)/32
      DO 300 I=2,CDELW
      STFBUF(I)=CDBUF(I)
  300 CONTINUE
      STFBIT=STFBIT+1
      KCNT=KCNT+1
      GO TO 340
  310 CONTINUE
C
C    1-D SHORTER
C
      CDELCT=TEMPEL
      CDDATA=TEMPCD
  320 CONTINUE
      KCNT=1
C
  340 CONTINUE
C
C    SAVE LINE LENGTH(DATA BITS ONLY)
C
      STAT(INLNCT)=CDDATA+CODERD(1,6)
C     SWITCH CODE & REFERENCE LINES
C
      TEMP=INREF
      INREF=INCOD
      INCOD=TEMP
  345 CONTINUE
C
C    ACCUMULATE STATISTICS AND ERROR CORRUPT
C
      IF(ERRMOD.EQ.NN) GO TO 390
C
C    ERROR CORRUPT
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C    ERROR IN RANGE OF CODED LINE; CHANGE APPROPRIATE BIT
C
      BIT=I4B(STFBUF,ERRBIT+32,1)
      BIT=MOD(BIT+1,2)
      CALL MI2B(BIT,STFBUF,ERRBIT+32,1)
      ERRCNT=ERRCNT+1
C
C    INCREMENT ERROR LIST POINTER
C
```

```
  360 CONTINUE
      ERRPNT=ERRPNT+1
      IF(ERRPNT.LE.ERRLIM) GO TO 350
C
C   ERROR LIST EXHAUSTED
C
      ERRPNT=ERRPNT-1
      WRITE(LPFIL,370) ERRPNT,ERRORS(ERRPNT)
  370 FORMAT('0ERROR LIST EXHAUSTED AT',I10,'TH ERROR;'/
     *       ' LAST ERROR OCCURRED AT',I10,' BITS')
      ERRMOD=NN
C
C   COMPUTE STATISTICS
C
  390 CONTINUE
      TCDEL=TCDEL+CDELCT-32
      TCDATA=TCDATA+CDDATA
      IF(DIAG) WRITE(TERM,160) INLNCT, CDDATA
C
      IF (.NOT.DIAG) GO TO 460
      CDELW=(CDELCT+32-1)/32
      WRITE(LPFIL,450) (CDBUF(I),I=1,CDELW)
      WRITE(LPFIL,450) (STFBUF(I),I=1,CDELW)
  450 FORMAT(6Z12)
  460 CONTINUE
      RETURN
C
  500 CONTINUE
      CALL EXIT
C
      E N D
      SUBROUTINE CODE3M(MODE,POLAR,A,B,C,CDELCT,CDDATA)
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C
C******************** BEGIN PROGRAM ****************************
C
      GO TO (100,100,100,200,100,800,800) ,MODE
C
C   MODE      1   2   3   4   5   6   7
C
      STOP 129
C
C   PASS MODE(5),VERTICAL MODE: A1B1=0(1), A1B1=1(2,3)
C
 100  CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      CDDATA=CDDATA+CODERD(1,MODE)
      RETURN
C
C   HORIZONTAL MODE(4)
C
  200 CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      CDDATA=CDDATA+CODERD(1,MODE)
      CALL CODELN(B-A,POLAR,CDELCT,CDDATA)
      NEWPOL=MOD(POLAR+2,2)+1
      CALL CODELN(C-B,NEWPOL,CDELCT,CDDATA)
      RETURN
C
C   ADD EOL1 OR EOL2 TO LINE (6,7)
C
 800  CONTINUE
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      RETURN
      END
      SUBROUTINE ONED3(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
```

```
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *           CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *           ERRCNT,INLNCT,CONSEC,LNNCBF,KCNT,
     *           INCOD,INREF,OTCOD,OTREF,STFBIT,VMMD
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODE(1,INDEX,COLOR)
      CALL  GETL3(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z8,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODE(3,INDEX,COLOR)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODE(2,INDEX,COLOR)
      IF(INDEX.GE.93) GO TO 1190
      IF(CODE(1,INDEX,COLOR).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
C
C    TEST FOR MAKE UP OR TERMINATING CODE
C
      RUNLEN=INDEX-1
      IF(INDEX.GE.65) RUNLEN=(INDEX-64)*64
      IF(RUNLEN.EQ.0) GO TO 1160
      IF(COLOR.EQ.1) GO TO 1155
      IF(RUNLEN.LT.0) STOP 1100
C
C    ADD BLACK RUN TO OUTPUT BUFFER
C
      DO 1150 I=1,RUNLEN
      CALL MI2B(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C    ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C OUTPUT LINE LESS THAN OR EQUAL TO MAX SPECIFIED
C
 1160 CONTINUE
      IF(INDEX.LT.65) GO TO 1170
      INDEX=3
      GO TO 1000
C
C    RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1170 CONTINUE
      CHCOL=.TRUE.
      STATUS=1
      RETURN
```

```
C
C    RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C    NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C    EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C    EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      SUBROUTINE TWOD3(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,KCNT,
     *            INCOD,INREF,OTCOD,OTREF,STFBIT,VMMD
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODERD(1,INDEX)
      CALL GETL3(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z12,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODERD(3,INDEX)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODERD(2,INDEX)
      IF(INDEX.GE.8) GO TO 1190
      IF(CODERD(1,INDEX).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
```

```
C
C    FIND B1 AND B2
C
      A0=OTELP
      IF(OTELP.EQ.1) A0=0
      POL=COLOR-1
C
C    DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 65
      PELM1=0
      IF(A0.EQ.0) GO TO 50
      PELM1=I4B(OTBUF(1,OTREF),A0,1)
   50 CONTINUE
      PEL=I4B(OTBUF(1,OTREF),I,1)
      IF(PEL.NE.PELM1) GO TO 70
   60 CONTINUE
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 50
   65 CONTINUE
      B1=I
      GO TO 92
   70 CONTINUE
      IF(PEL.NE.POL) GO TO 90
      GO TO 60
   90 CONTINUE
      B1=I
   92 CONTINUE
      GO TO (300,400,600,200,100),INDEX
      STOP 100
C
C    PASS MODE
C
  100 CONTINUE
      RUNLEN=B2-OTELP
      CHCOL=.FALSE.
      GO TO (1155,1145),COLOR
C
C    HORIZONTAL MODE
C
  200 CONTINUE
      ENTRY=3
      CALL ONED3(ENTRY,COLOR,STATE,L)
      GO TO (210,1180,1190,1200,1205),STATE
  210 CONTINUE
      COLOR=MOD(COLOR+2,2)+1
      ENTRY=3
      CALL ONED3(ENTRY,COLOR,STATE,L)
      GO TO (220,1180,1190,1200,1205),STATE
  220 CONTINUE
      CHCOL=.TRUE.
      GO TO 1160
C
C    VERTICAL MODE A1B1=0
C
  300 CONTINUE
      RUNLEN=B1-OTELP
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C    VERTICAL MODE CORRELATED (VMC)
C
  400 CONTINUE
      IF(VMMD) 410,420,430
  410 STOP 410
  420 CONTINUE
      RUNLEN=B1-OTELP+1
       VMMD=0
      GO TO 440
  430 CONTINUE
      RUNLEN=B1-OTELP-1
      VMMD=1
  440 CONTINUE
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C    VERTICAL MODE UNCORRELATED (VMU)
C
  600 CONTINUE
      IF(VMMD) 610,620,630
```

```
  610 STOP 610
  620 CONTINUE
      RUNLEN=81-OTELP-1
      VMMD=1
      GO TO 640
  630 CONTINUE
      RUNLEN=81-OTELP+1
      VMMD=0
  640 CONTINUE
      CHCOL=.TRUE.
      GO TO (1155,1145), COLOR
C
C    ADD BLACK RUN TO OUTPUT BUFFER
C
 1145 CONTINUE
      IF(RUNLEN) 1190,1160,1147
 1147 CONTINUE
      DO 1150 I=1,RUNLEN
      CALL MI28(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C    ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      IF(RUNLEN.LT.0) GO TO 1190
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C    RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1160 CONTINUE
      STATUS=1
      RETURN
C
C    RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C    NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C    EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C    EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      BLOCK DATA
C
      IMPLICIT INTEGER(A-Z)
C************************ FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,ONECNT,LNNOBF,WRCBUF,LPACK,
     *            INCOD,INREF,OTCOD,OTREF,TSTFBT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
```

```
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
C
      DATA TERM,LPFIL,PELFIL,OTFIL,ERFIL/5,6,1,2,3/
      DATA DD,II,MM,TT,NN,YY/'D','I','M','T','N','Y'/
      DATA PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX/1728,2,0,96,'T',3000/
      DATA K/2/
      DATA DIAG/.FALSE./
C
       DATA CODE(1,  1,1),CODE(2,  1,1),CODE(3,  1,1)/ 8, 70,Z0035/
       DATA CODE(1,  2,1),CODE(2,  2,1),CODE(3,  2,1)/ 6, 90,Z00C7/
       DATA CODE(1,  3,1),CODE(2,  3,1),CODE(3,  3,1)/ 4,  4,Z0007/
       DATA CODE(1,  4,1),CODE(2,  4,1),CODE(3,  4,1)/ 4,  5,Z0006/
       DATA CODE(1,  5,1),CODE(2,  5,1),CODE(3,  5,1)/ 4,  6,Z0008/
       DATA CODE(1,  6,1),CODE(2,  6,1),CODE(3,  6,1)/ 4,  7,Z000C/
       DATA CODE(1,  7,1),CODE(2,  7,1),CODE(3,  7,1)/ 4,  8,Z000E/
       DATA CODE(1,  8,1),CODE(2,  8,1),CODE(3,  8,1)/ 4,  9,Z000F/
       DATA CODE(1,  9,1),CODE(2,  9,1),CODE(3,  9,1)/ 5, 10,Z0013/
       DATA CODE(1, 10,1),CODE(2, 10,1),CODE(3, 10,1)/ 5, 11,Z0014/
       DATA CODE(1, 11,1),CODE(2, 11,1),CODE(3, 11,1)/ 5, 12,Z0007/
       DATA CODE(1, 12,1),CODE(2, 12,1),CODE(3, 12,1)/ 5, 65,Z0008/
       DATA CODE(1, 13,1),CODE(2, 13,1),CODE(3, 13,1)/ 6, 14,Z0 38/
       DATA CODE(1, 14,1),CODE(2, 14,1),CODE(3, 14,1)/ 6, 15,Z0003/
       DATA CODE(1, 15,1),CODE(2, 15,1),CODE(3, 15,1)/ 6, 16,Z0034/
       DATA CODE(1, 16,1),CODE(2, 16,1),CODE(3, 16,1)/ 6, 17,Z0035/
       DATA CODE(1, 17,1),CODE(2, 17,1),CODE(3, 17,1)/ 6, 18,Z002A/
       DATA CODE(1, 18,1),CODE(2, 18,1),CODE(3, 18,1)/ 6, 19,Z002B/
       DATA CODE(1, 19,1),CODE(2, 19,1),CODE(3, 19,1)/ 7, 20,Z0027/
       DATA CODE(1, 20,1),CODE(2, 20,1),CODE(3, 20,1)/ 7, 21,Z00CC/
       DATA CODE(1, 21,1),CODE(2, 21,1),CODE(3, 21,1)/ 7, 22,Z0008/
       DATA CODE(1, 22,1),CODE(2, 22,1),CODE(3, 22,1)/ 7, 23,Z0017/
       DATA CODE(1, 23,1),CODE(2, 23,1),CODE(3, 23,1)/ 7, 24,Z0003/
       DATA CODE(1, 24,1),CODE(2, 24,1),CODE(3, 24,1)/ 7, 25,Z0004/
       DATA CODE(1, 25,1),CODE(2, 25,1),CODE(3, 25,1)/ 7, 26,Z0028/
       DATA CODE(1, 26,1),CODE(2, 26,1),CODE(3, 26,1)/ 7, 27,Z002B/
       DATA CODE(1, 27,1),CODE(2, 27,1),CODE(3, 27,1)/ 7, 28,Z0013/
       DATA CODE(1, 28,1),CODE(2, 28,1),CODE(3, 28,1)/ 7, 29,Z0024/
       DATA CODE(1, 29,1),CODE(2, 29,1),CODE(3, 29,1)/ 7, 68,Z0018/
       DATA CODE(1, 30,1),CODE(2, 30,1),CODE(3, 30,1)/ 8, 31,Z0002/
       DATA CODE(1, 31,1),CODE(2, 31,1),CODE(3, 31,1)/ 8, 32,Z0003/
       DATA CODE(1, 32,1),CODE(2, 32,1),CODE(3, 32,1)/ 8, 33,Z001A/
       DATA CODE(1, 33,1),CODE(2, 33,1),CODE(3, 33,1)/ 8, 34,Z001B/
       DATA CODE(1, 34,1),CODE(2, 34,1),CODE(3, 34,1)/ 8, 35,Z0012/
       DATA CODE(1, 35,1),CODE(2, 35,1),CODE(3, 35,1)/ 8, 36,Z0013/
       DATA CODE(1, 36,1),CODE(2, 36,1),CODE(3, 36,1)/ 8, 37,Z0014/
       DATA CODE(1, 37,1),CODE(2, 37,1),CODE(3, 37,1)/ 8, 38,Z0015/
       DATA CODE(1, 38,1),CODE(2, 38,1),CODE(3, 38,1)/ 8, 39,Z0016/
       DATA CODE(1, 39,1),CODE(2, 39,1),CODE(3, 39,1)/ 8, 40,Z0017/
       DATA CODE(1, 40,1),CODE(2, 40,1),CODE(3, 40,1)/ 8, 41,Z0028/
       DATA CODE(1, 41,1),CODE(2, 41,1),CODE(3, 41,1)/ 8, 42,Z0029/
       DATA CODE(1, 42,1),CODE(2, 42,1),CODE(3, 42,1)/ 8, 43,Z002A/
       DATA CODE(1, 43,1),CODE(2, 43,1),CODE(3, 43,1)/ 8, 44,Z002B/
       DATA CODE(1, 44,1),CODE(2, 44,1),CODE(3, 44,1)/ 8, 45,Z002C/
       DATA CODE(1, 45,1),CODE(2, 45,1),CODE(3, 45,1)/ 8, 46,Z002D/
       DATA CODE(1, 46,1),CODE(2, 46,1),CODE(3, 46,1)/ 8, 47,Z0004/
       DATA CODE(1, 47,1),CODE(2, 47,1),CODE(3, 47,1)/ 8, 48,Z0005/
       DATA CODE(1, 48,1),CODE(2, 48,1),CODE(3, 48,1)/ 8, 49,Z000A/
       DATA CODE(1, 49,1),CODE(2, 49,1),CODE(3, 49,1)/ 8, 50,Z000B/
       DATA CODE(1, 50,1),CODE(2, 50,1),CODE(3, 50,1)/ 8, 51,Z0052/
       DATA CODE(1, 51,1),CODE(2, 51,1),CODE(3, 51,1)/ 8, 52,Z0053/
       DATA CODE(1, 52,1),CODE(2, 52,1),CODE(3, 52,1)/ 8, 53,Z0054/
       DATA CODE(1, 53,1),CODE(2, 53,1),CODE(3, 53,1)/ 8, 54,Z0055/
       DATA CODE(1, 54,1),CODE(2, 54,1),CODE(3, 54,1)/ 8, 55,Z0024/
       DATA CODE(1, 55,1),CODE(2, 55,1),CODE(3, 55,1)/ 8, 56,Z0025/
       DATA CODE(1, 56,1),CODE(2, 56,1),CODE(3, 56,1)/ 8, 57,Z0058/
       DATA CODE(1, 57,1),CODE(2, 57,1),CODE(3, 57,1)/ 8, 58,Z0059/
       DATA CODE(1, 58,1),CODE(2, 58,1),CODE(3, 58,1)/ 8, 59,Z005A/
       DATA CODE(1, 59,1),CODE(2, 59,1),CODE(3, 59,1)/ 8, 60,Z005B/
       DATA CODE(1, 60,1),CODE(2, 60,1),CODE(3, 60,1)/ 8, 61,Z004A/
       DATA CODE(1, 61,1),CODE(2, 61,1),CODE(3, 61,1)/ 8, 62,Z004B/
       DATA CODE(1, 62,1),CODE(2, 62,1),CODE(3, 62,1)/ 8, 63,Z0032/
       DATA CODE(1, 63,1),CODE(2, 63,1),CODE(3, 63,1)/ 8, 64,Z0033/
       DATA CODE(1, 64,1),CODE(2, 64,1),CODE(3, 64,1)/ 8, 69,Z0034/
       DATA CODE(1, 65,1),CODE(2, 65,1),CODE(3, 65,1)/ 5, 66,Z001B/
       DATA CODE(1, 66,1),CODE(2, 66,1),CODE(3, 66,1)/ 5, 67,Z0012/
       DATA CODE(1, 67,1),CODE(2, 67,1),CODE(3, 67,1)/ 6,  2,Z0017/
       DATA CODE(1, 68,1),CODE(2, 68,1),CODE(3, 68,1)/ 7, 30,Z0037/
       DATA CODE(1, 69,1),CODE(2, 69,1),CODE(3, 69,1)/ 8,  1,Z0036/
       DATA CODE(1, 70,1),CODE(2, 70,1),CODE(3, 70,1)/ 8, 71,Z0037/
       DATA CODE(1, 71,1),CODE(2, 71,1),CODE(3, 71,1)/ 9, 72,Z0064/
       DATA CODE(1, 72,1),CODE(2, 72,1),CODE(3, 72,1)/ 8, 73,Z0065/
       DATA CODE(1, 73,1),CODE(2, 73,1),CODE(3, 73,1)/ 8, 74,Z0068/
```

```
DATA CODE(1, 74,1),CODE(2, 74,1),CODE(3, 74,1)/ 8, 75,Z0067/
DATA CODE(1, 75,1),CODE(2, 75,1),CODE(3, 75,1)/ 9, 76,Z00CC/
DATA CODE(1, 76,1),CODE(2, 76,1),CODE(3, 76,1)/ 9, 77,Z00CD/
DATA CODE(1, 77,1),CODE(2, 77,1),CODE(3, 77,1)/ 9, 78,Z00D2/
DATA CODE(1, 78,1),CODE(2, 78,1),CODE(3, 78,1)/ 9, 79,Z00D3/
DATA CODE(1, 79,1),CODE(2, 79,1),CODE(3, 79,1)/ 9, 80,Z00D4/
DATA CODE(1, 80,1),CODE(2, 80,1),CODE(3, 80,1)/ 9, 81,Z00D5/
DATA CODE(1, 81,1),CODE(2, 81,1),CODE(3, 81,1)/ 9, 82,Z00D6/
DATA CODE(1, 82,1),CODE(2, 82,1),CODE(3, 82,1)/ 9, 83,Z00D7/
DATA CODE(1, 83,1),CODE(2, 83,1),CODE(3, 83,1)/ 9, 84,Z00D8/
DATA CODE(1, 84,1),CODE(2, 84,1),CODE(3, 84,1)/ 9, 85,Z00D9/
DATA CODE(1, 85,1),CODE(2, 85,1),CODE(3, 85,1)/ 9, 86,Z00DA/
DATA CODE(1, 86,1),CODE(2, 86,1),CODE(3, 86,1)/ 9, 87,Z00DB/
DATA CODE(1, 87,1),CODE(2, 87,1),CODE(3, 87,1)/ 9, 88,Z0098/
DATA CODE(1, 88,1),CODE(2, 88,1),CODE(3, 88,1)/ 9, 89,Z0099/
DATA CODE(1, 89,1),CODE(2, 89,1),CODE(3, 89,1)/ 9, 91,Z009A/
DATA CODE(1, 90,1),CODE(2, 90,1),CODE(3, 90,1)/ 6, 13,Z0018/
DATA CODE(1, 91,1),CODE(2, 91,1),CODE(3, 91,1)/ 9, 92,Z009B/
DATA CODE(1, 92,1),CODE(2, 92,1),CODE(3, 92,1)/13, 93,Z0002/
DATA CODE(1,  1,2),CODE(2,  1,2),CODE(3,  1,2)/10, 65,Z0037/
DATA CODE(1,  2,2),CODE(2,  2,2),CODE(3,  2,2)/ 3,  6,Z0C02/
DATA CODE(1,  3,2),CODE(2,  3,2),CODE(3,  3,2)/ 2,  4,Z0003/
DATA CODE(1,  4,2),CODE(2,  4,2),CODE(3,  4,2)/ 2,  5,Z0002/
DATA CODE(1,  5,2),CODE(2,  5,2),CODE(3,  5,2)/ 3,  2,Z0003/
DATA CODE(1,  6,2),CODE(2,  6,2),CODE(3,  6,2)/ 4,  7,Z0003/
DATA CODE(1,  7,2),CODE(2,  7,2),CODE(3,  7,2)/ 4,  8,Z0002/
DATA CODE(1,  8,2),CODE(2,  8,2),CODE(3,  8,2)/ 5,  9,Z0003/
DATA CODE(1,  9,2),CODE(2,  9,2),CODE(3,  9,2)/ 6, 10,Z0005/
DATA CODE(1, 10,2),CODE(2, 10,2),CODE(3, 10,2)/ 6, 11,Z0004/
DATA CODE(1, 11,2),CODE(2, 11,2),CODE(3, 11,2)/ 7, 12,Z0004/
DATA CODE(1, 12,2),CODE(2, 12,2),CODE(3, 12,2)/ 7, 13,Z0005/
DATA CODE(1, 13,2),CODE(2, 13,2),CODE(3, 13,2)/ 7, 14,Z0007/
DATA CODE(1, 14,2),CODE(2, 14,2),CODE(3, 14,2)/ 8, 15,Z0004/
DATA CODE(1, 15,2),CODE(2, 15,2),CODE(3, 15,2)/ 8, 16,Z0007/
DATA CODE(1, 16,2),CODE(2, 16,2),CODE(3, 16,2)/ 9, 17,Z0018/
DATA CODE(1, 17,2),CODE(2, 17,2),CODE(3, 17,2)/10, 18,Z0017/
DATA CODE(1, 18,2),CODE(2, 18,2),CODE(3, 18,2)/10, 19,Z0018/
DATA CODE(1, 19,2),CODE(2, 19,2),CODE(3, 19,2)/10,  1,Z0008/
DATA CODE(1, 20,2),CODE(2, 20,2),CODE(3, 20,2)/11, 21,Z0067/
DATA CODE(1, 21,2),CODE(2, 21,2),CODE(3, 21,2)/11, 22,Z0068/
DATA CODE(1, 22,2),CODE(2, 22,2),CODE(3, 22,2)/11, 23,Z006C/
DATA CODE(1, 23,2),CODE(2, 23,2),CODE(3, 23,2)/11, 24,Z0037/
DATA CODE(1, 24,2),CODE(2, 24,2),CODE(3, 24,2)/11, 25,Z0028/
DATA CODE(1, 25,2),CODE(2, 25,2),CODE(3, 25,2)/11, 26,Z0017/
DATA CODE(1, 26,2),CODE(2, 26,2),CODE(3, 26,2)/11, 27,Z0018/
DATA CODE(1, 27,2),CODE(2, 27,2),CODE(3, 27,2)/12, 28,Z00CA/
DATA CODE(1, 28,2),CODE(2, 28,2),CODE(3, 28,2)/12, 29,Z00CB/
DATA CODE(1, 29,2),CODE(2, 29,2),CODE(3, 29,2)/12, 30,Z00CC/
DATA CODE(1, 30,2),CODE(2, 30,2),CODE(3, 30,2)/12, 31,Z00CD/
DATA CODE(1, 31,2),CODE(2, 31,2),CODE(3, 31,2)/12, 32,Z0068/
DATA CODE(1, 32,2),CODE(2, 32,2),CODE(3, 32,2)/12, 33,Z0069/
DATA CODE(1, 33,2),CODE(2, 33,2),CODE(3, 33,2)/12, 34,Z006A/
DATA CODE(1, 34,2),CODE(2, 34,2),CODE(3, 34,2)/12, 35,Z006B/
DATA CODE(1, 35,2),CODE(2, 35,2),CODE(3, 35,2)/12, 36,Z00D2/
DATA CODE(1, 36,2),CODE(2, 36,2),CODE(3, 36,2)/12, 37,Z00D3/
DATA CODE(1, 37,2),CODE(2, 37,2),CODE(3, 37,2)/12, 38,Z00D4/
DATA CODE(1, 38,2),CODE(2, 38,2),CODE(3, 38,2)/12, 39,Z00D5/
DATA CODE(1, 39,2),CODE(2, 39,2),CODE(3, 39,2)/12, 40,Z00D6/
DATA CODE(1, 40,2),CODE(2, 40,2),CODE(3, 40,2)/12, 41,Z00D7/
DATA CODE(1, 41,2),CODE(2, 41,2),CODE(3, 41,2)/12, 42,Z006C/
DATA CODE(1, 42,2),CODE(2, 42,2),CODE(3, 42,2)/12, 43,Z006D/
DATA CODE(1, 43,2),CODE(2, 43,2),CODE(3, 43,2)/12, 44,Z00DA/
DATA CODE(1, 44,2),CODE(2, 44,2),CODE(3, 44,2)/12, 45,Z00DB/
DATA CODE(1, 45,2),CODE(2, 45,2),CODE(3, 45,2)/12, 46,Z0054/
DATA CODE(1, 46,2),CODE(2, 46,2),CODE(3, 46,2)/12, 47,Z0055/
DATA CODE(1, 47,2),CODE(2, 47,2),CODE(3, 47,2)/12, 48,Z0056/
DATA CODE(1, 48,2),CODE(2, 48,2),CODE(3, 48,2)/12, 49,Z0057/
DATA CODE(1, 49,2),CODE(2, 49,2),CODE(3, 49,2)/12, 50,Z0064/
DATA CODE(1, 50,2),CODE(2, 50,2),CODE(3, 50,2)/12, 51,Z0065/
DATA CODE(1, 51,2),CODE(2, 51,2),CODE(3, 51,2)/12, 52,Z0052/
DATA CODE(1, 52,2),CODE(2, 52,2),CODE(3, 52,2)/12, 53,Z0053/
DATA CODE(1, 53,2),CODE(2, 53,2),CODE(3, 53,2)/12, 54,Z0024/
DATA CODE(1, 54,2),CODE(2, 54,2),CODE(3, 54,2)/12, 55,Z0037/
DATA CODE(1, 55,2),CODE(2, 55,2),CODE(3, 55,2)/12, 56,Z0038/
DATA CODE(1, 56,2),CODE(2, 56,2),CODE(3, 56,2)/12, 57,Z0027/
DATA CODE(1, 57,2),CODE(2, 57,2),CODE(3, 57,2)/12, 58,Z0028/
DATA CODE(1, 58,2),CODE(2, 58,2),CODE(3, 58,2)/12, 59,Z0058/
DATA CODE(1, 59,2),CODE(2, 59,2),CODE(3, 59,2)/12, 60,Z0059/
DATA CODE(1, 60,2),CODE(2, 60,2),CODE(3, 60,2)/12, 61,Z002B/
DATA CODE(1, 61,2),CODE(2, 61,2),CODE(3, 61,2)/12, 62,Z002C/
DATA CODE(1, 62,2),CODE(2, 62,2),CODE(3, 62,2)/12, 63,Z005A/
DATA CODE(1, 63,2),CODE(2, 63,2),CODE(3, 63,2)/12, 64,Z0066/
```

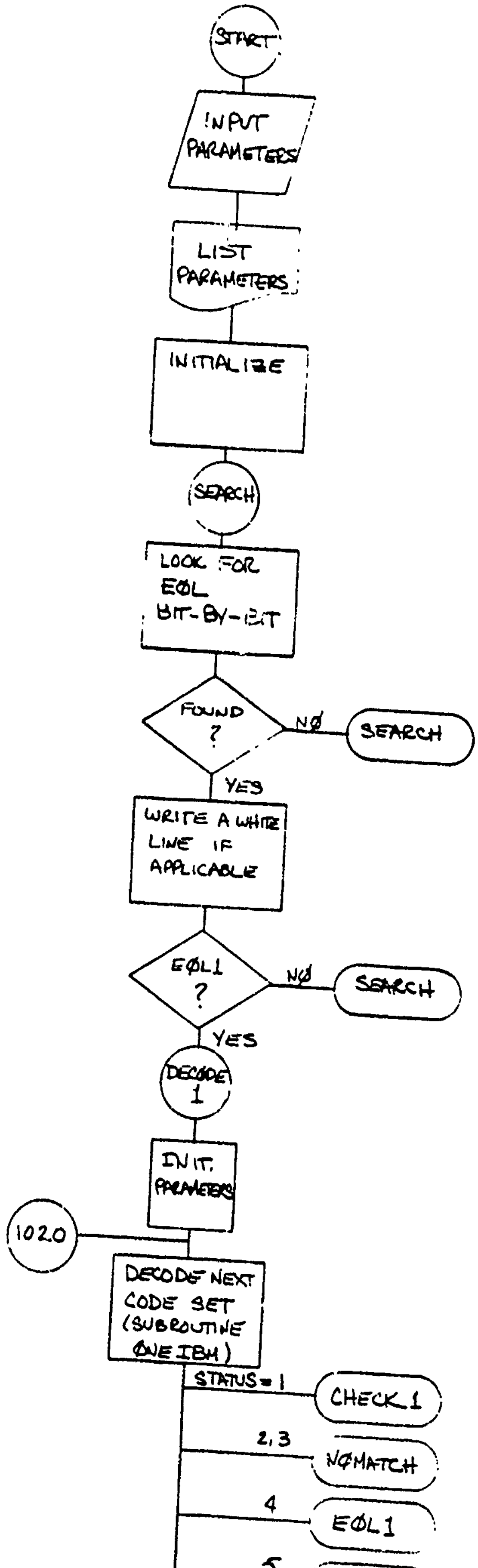
```
      DATA CODE(1, 64,2),CODE(2, 64,2),CODE(3, 64,2)/12, 66,Z0067/
      DATA CODE(1, 65,2),CODE(2, 65,2),CODE(3, 65,2)/10, 20,Z000F/
      DATA CODE(1, 66,2),CODE(2, 66,2),CODE(3, 66,2)/12, 67,Z00C8/
      DATA CODE(1, 67,2),CODE(2, 67,2),CODE(3, 67,2)/12, 68,Z00C9/
      DATA CODE(1, 68,2),CODE(2, 68,2),CODE(3, 68,2)/12, 69,Z0058/
      DATA CODE(1, 69,2),CODE(2, 69,2),CODE(3, 69,2)/12, 70,Z0033/
      DATA CODE(1, 70,2),CODE(2, 70,2),CODE(3, 70,2)/12, 71,Z0034/
      DATA CODE(1, 71,2),CODE(2, 71,2),CODE(3, 71,2)/12, 72,Z0035/
      DATA CODE(1, 72,2),CODE(2, 72,2),CODE(3, 72,2)/13, 73,Z006C/
      DATA CODE(1, 73,2),CODE(2, 73,2),CODE(3, 73,2)/13, 74,Z006D/
      DATA CODE(1, 74,2),CODE(2, 74,2),CODE(3, 74,2)/13, 75,Z004A/
      DATA CODE(1, 75,2),CODE(2, 75,2),CODE(3, 75,2)/13, 76,Z004B/
      DATA CODE(1, 76,2),CODE(2, 76,2),CODE(3, 76,2)/13, 77,Z004C/
      DATA CODE(1, 77,2),CODE(2, 77,2),CODE(3, 77,2)/13, 78,Z004D/
      DATA CODE(1, 78,2),CODE(2, 78,2),CODE(3, 78,2)/13, 79,Z0072/
      DATA CODE(1, 79,2),CODE(2, 79,2),CODE(3, 79,2)/13, 80,Z0073/
      DATA CODE(1, 80,2),CODE(2, 80,2),CODE(3, 80,2)/13, 81,Z0074/
      DATA CODE(1, 81,2),CODE(2, 81,2),CODE(3, 81,2)/13, 82,Z0075/
      DATA CODE(1, 82,2),CODE(2, 82,2),CODE(3, 82,2)/13, 83,Z0076/
      DATA CODE(1, 83,2),CODE(2, 83,2),CODE(3, 83,2)/13, 84,Z0077/
      DATA CODE(1, 84,2),CODE(2, 84,2),CODE(3, 84,2)/13, 85,Z0052/
      DATA CODE(1, 85,2),CODE(2, 85,2),CODE(3, 85,2)/13, 86,Z0053/
      DATA CODE(1, 86,2),CODE(2, 86,2),CODE(3, 86,2)/13, 87,Z0054/
      DATA CODE(1, 87,2),CODE(2, 87,2),CODE(3, 87,2)/13, 88,Z0055/
      DATA CODE(1, 88,2),CODE(2, 88,2),CODE(3, 88,2)/13, 89,Z005A/
      DATA CODE(1, 89,2),CODE(2, 89,2),CODE(3, 89,2)/13, 90,Z005B/
      DATA CODE(1, 90,2),CODE(2, 90,2),CODE(3, 90,2)/13, 91,Z0064/
      DATA CODE(1, 91,2),CODE(2, 91,2),CODE(3, 91,2)/13, 92,Z0065/
      DATA CODE(1, 92,2),CODE(2, 92,2),CODE(3, 92,2)/13, 93,Z0003/
      DATA CODERD(1,1),CODERD(2,1),CODERD(3,1)/        1,2,Z1/
      DATA CODERD(1,2),CODERD(2,2),CODERD(3,2)/        2,3,Z1/
      DATA CODERD(1,3),CODERD(2,3),CODERD(3,3)/        3,4,Z1/
      DATA CODERD(1,4),CODERD(2,4),CODERD(3,4)/        4,6,Z1/
      DATA CODERD(1,5),CODERD(2,5),CODERD(3,5)/        5,0,Z1/
      DATA CODERD(1,6),CODERD(2,6),CODERD(3,6)/        13,7,Z2/
      DATA CODERD(1,7),CODERD(2,7),CODERD(3,7)/        13,8,Z3/
C
      END
0          END OF OCEC UPRINT PROGRAM - -      - ---- LINES PRINTED=  1431
```

APPENDIX K

PROGRAM FLOW CHART

FOR IBM ALGORITHM

IBM

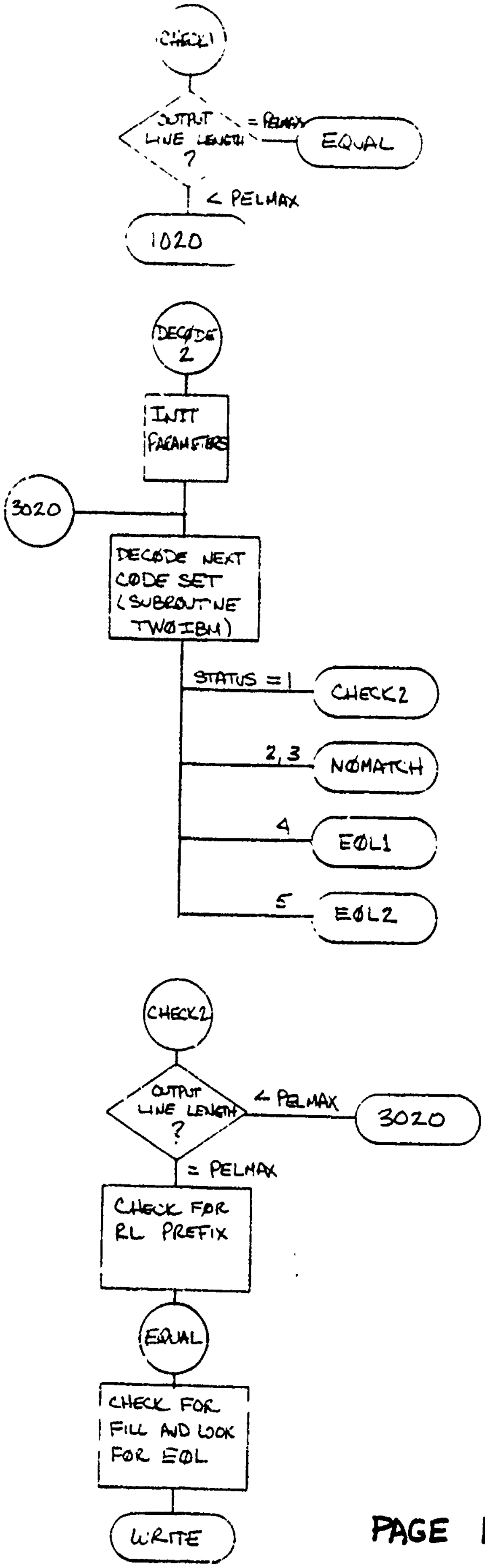


ØUTPUT LINE LENGTH ≤ PELMAX

ØUTPUT LINE TØØ LONG ØR NØ MATCH FØUND IN CODE TABLE

PREMATURE EØL DETECTED

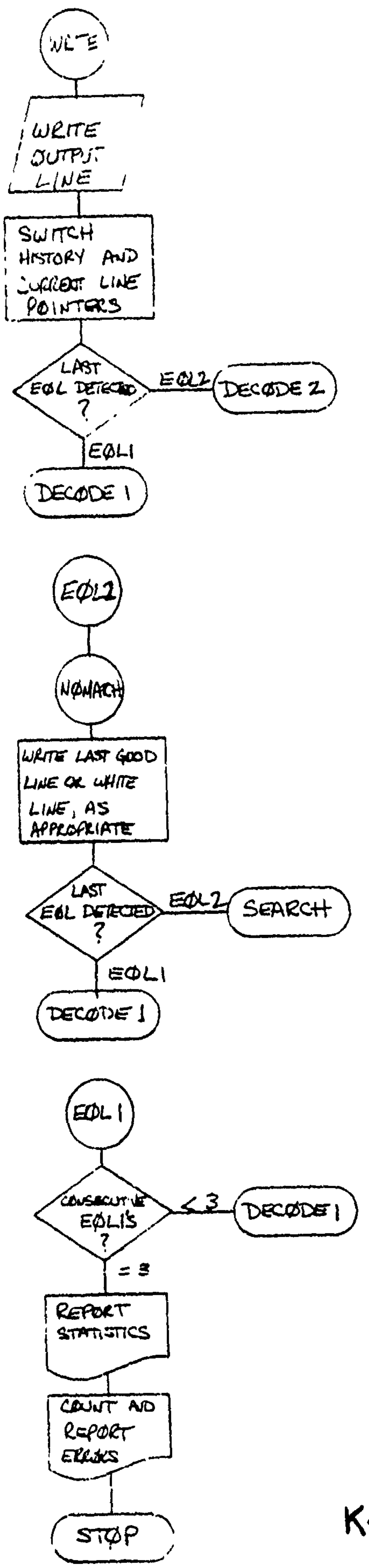
=2.8



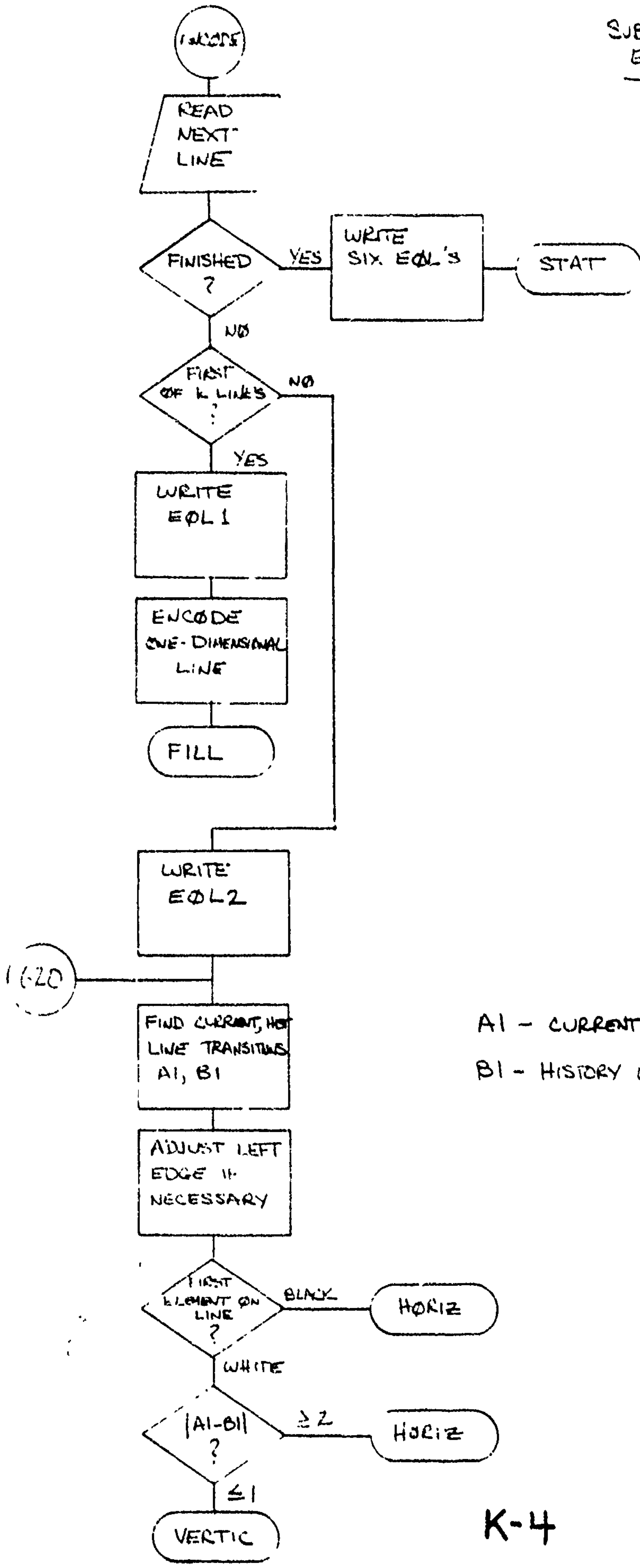
ERROR CONDITIONS DETECTED:

1) NØ RL PREFIX

2) < 11 ZERØES IN EØL

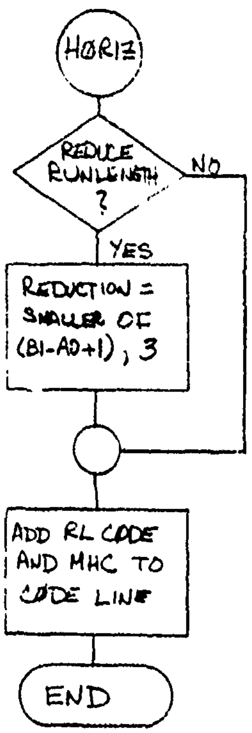

WRITE
WRITE OUTPUT LINE
SWITCH HISTORY AND CURRENT LINE POINTERS
LAST EØL DETECTED ?
EØL2
DECØDE 2
EØL1
DECØDE 1
EØL2
NØMATCH
WRITE LAST GOOD LINE OR WHITE LINE, AS APPROPRIATE
LAST EØL DETECTED ?
EØL2
SEARCH
EØL1
DECØDE 1
EØL1
CONSECUTIVE EØL1'S ?
< 3
DECØDE 1
= 3
REPORT STATISTICS
COUNT AND REPORT ERRORS
STØP

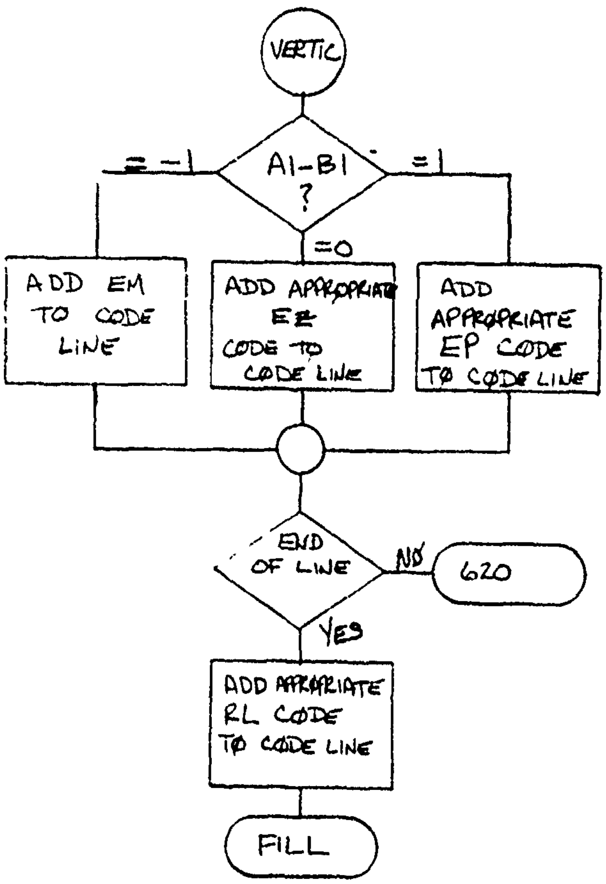
SUBROUTINE ENCODE

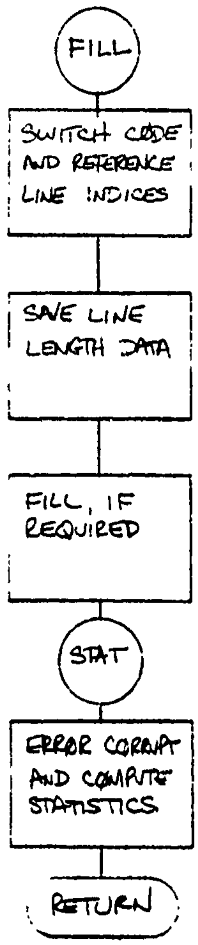


A1 – CURRENT LINE TRANSITION ELEMENT

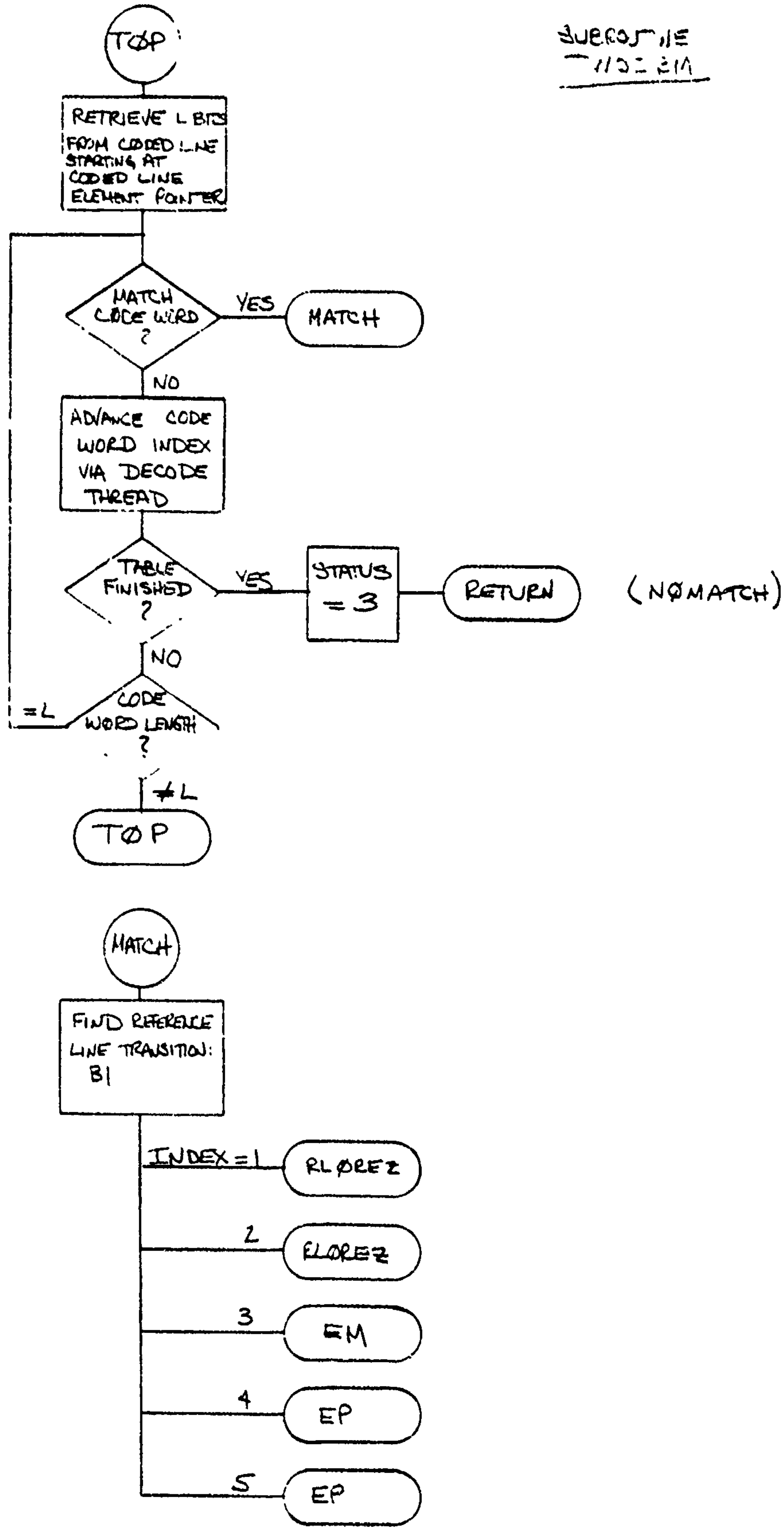
B1 – HISTORY LINE TRANSITION ELEMENT

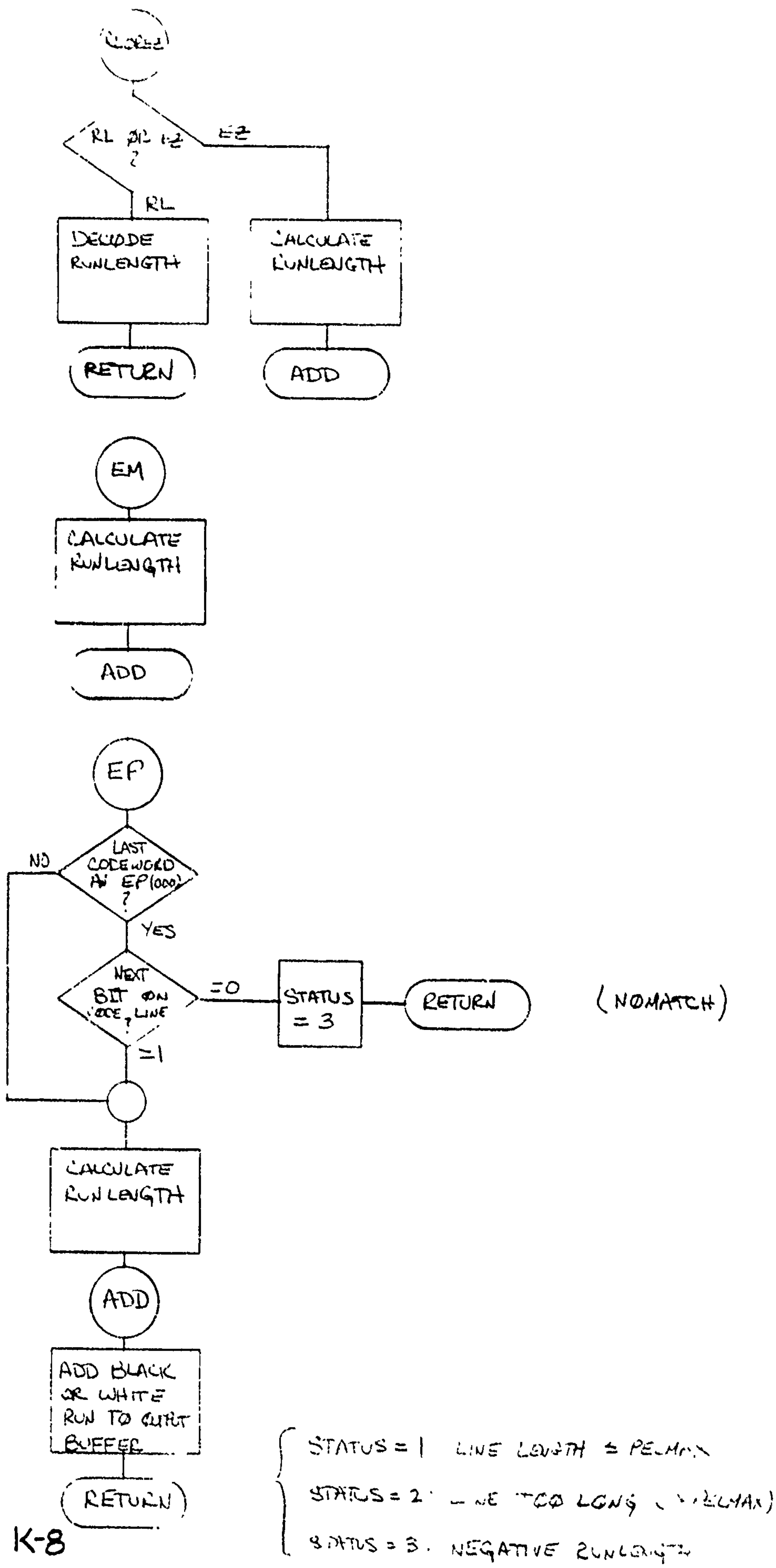

HØRIZ
REDUCE RUNLENGTH ?
NO
YES
REDUCTION = SMALLER OF (B1-A0+1), 3
ADD RL CØDE AND MHC TO CØDE LINE
END


VERTIC
A1-B1 ?
= -1
= 1
=0
ADD EM TO CODE LINE
ADD APPROPRIATE EZ CODE TO CØDE LINE
ADD APPRØPRIATE EP CØDE TØ CØDE LINE
END OF LINE
NO
620
YES
ADD APPRØPRIATE RL CØDE TØ CØDE LINE
FILL


FILL
SWITCH CODE AND REFERENCE LINE INDICES
SAVE LINE LENGTH DATA
FILL, IF REQUIRED
STAT
ERROR CORRUPT AND COMPUTE STATISTICS
RETURN

SUBROUTINE


TØP
RETRIEVE L BITS FROM CODED LINE STARTING AT CODED LINE ELEMENT POINTER
MATCH CODE WORD ?
YES
MATCH
NO
ADVANCE CODE WORD INDEX VIA DECODE THREAD
TABLE FINISHED ?
YES
STATUS = 3
RETURN
(NØMATCH)
NO
CODE WORD LENGTH ?
=L
≠L
TØP
MATCH
FIND REFERENCE LINE TRANSITION: B1
INDEX = 1
RLØREZ
2
ELØREZ
3
EM
4
EP
5
EP


CODE
RL OR EZ ?
EZ
RL
DECODE RUNLENGTH
CALCULATE RUNLENGTH
RETURN
ADD
EM
CALCULATE RUNLENGTH
ADD
EP
LAST CODEWORD AN EP(000) ?
NO
YES
NEXT BIT ON CODE LINE ?
=0
=1
STATUS = 3
RETURN
(NOMATCH)
CALCULATE RUNLENGTH
ADD
ADD BLACK OR WHITE RUN TO OUTPUT BUFFER
RETURN
STATUS = 1 LINE LENGTH ≤ PELMAX
STATUS = 2 LINE TOO LONG (> PELMAX)
STATUS = 3. NEGATIVE RUNLENGTH

APPENDIX L

COMPUTER PROGRAM CODE LISTING

IBM ALGORITHM

```
START OF OCEC UPRINT PROGRAM          DSNAME=D0031.IBM.FORT
C       PROGRAM IBM
        IMPLICIT INTEGER(A-Z)
        REAL CF3,CF4,ERRATE
C******* LABELED COMMON /G32BIT/ *******
C
        COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
        INTEGER MASK,COMASK,LIBIT,LZBIT
C
        COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *               STFBUF(240), STAT(3000)
        COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
        COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
        COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES *********************
C
        COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
        COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNOBF,BICNT,
     *              INCOD,INREF,OTCOD,OTREF,STFBIT
        COMMON/ICHAR/DD,II,MM,TT,NN,YY
        COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
        LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,RLFLAG,EPFLAG
C
C     READ INPUT PARAMETERS
   90 WRITE(TERM,100)
  100 FORMAT('$PARAMETERS: INPUT(=I), OR DEFAULT(=D)?')
      READ(TERM,110,ERR=90) INSW
  110 FORMAT(A1)
      IF (INSW.EQ.DD) GO TO 315
      IF (INSW.NE.II) GO TO 90
C
C     READ DIAGNOSTIC SWITCH
C
  114 WRITE(TERM,115)
  115 FORMAT('$DIAGNOSTIC PRINTOUT? (Y OR N): ')
      READ(TERM,110) INSW
      IF(INSW.EQ.YY) GO TO 116
      IF(INSW.EQ.NN) GO TO 120
      GO TO 114
  116 CONTINUE
      DIAG=.TRUE.
C
C     READ MAXIMUM NUMBER OF PELS PER LINE
C
  120 CONTINUE
      WRITE(TERM,130)
  130 FORMAT('$ENTER MAXIMUM NUMBER OF PELS PER LINE: ')
      READ(TERM,140,ERR=120) PELMAX
  140 FORMAT(I4)
      IF(PELMAX.GE.1.AND.PELMAX.LE.1728) GO TO 160
      WRITE(TERM,150) PELMAX
  150 FORMAT('0NUMBER OUT OF RANGE (=',I6,')')
      GO TO 120
C
C     READ VERTICAL SAMPLING
C
  160 CONTINUE
      WRITE(TERM,170)
  170 FORMAT('$ENTER VERTICAL SAMPLING: ')
      READ(TERM,180,ERR=160) VRES
  180 FORMAT(I2)
      IF(VRES.GE.1.AND.VRES.LE.10) GO TO 190
      WRITE(TERM,150) VRES
      GO TO 160
C
C     READ PARAMETER K
C
  190 CONTINUE
      WRITE(TERM,192)
  192 FORMAT('$ENTER PARAMETER K: ')
      READ(TERM,140,ERR=190) K
      IF(K.GE.1.AND.K.LE.3000) GO TO 200
      WRITE(TERM,150) K
      GO TO 190
C
C     READ ERROR PATTERN PHASE
```

```
C
  200 CONTINUE
      WRITE(TERM,210)
  210 FORMAT('$ENTER ERROR PATTERN PHASE: ')
      READ(TERM,220,ERR=200) EPHASE
  220 FORMAT(I1)
      IF(EPHASE.GE.0.AND.EPHASE.LE.3) GO TO 240
      WRITE(TERM,150) EPHASE
      GO TO 200
C
C   READ MINIMUM COMPRESSED LINE LENGTH
C
  240 CONTINUE
      WRITE(TERM,250)
  250 FORMAT('$ENTER MINIMUM COMPRESSED LINE LENGTH: ')
      READ(TERM,140,ERR=240) CMPMAX
      IF(CMPMAX.GE.0.AND.CMPMAX.LE.1728) GO TO 320
      WRITE(TERM,150) CMPMAX
      GO TO 240
C
C   READ NUMBER OF SCAN LINES TO BE PROCESSED
  320 CONTINUE
      WRITE(TERM,330)
  330 FORMAT('$NUMBER OF SCAN LINES TO BE PROCESSED=? ')
      READ(TERM,140,ERR=320) LINMAX
      IF(LINMAX.GE.1.AND.LINMAX.LE.3000) GO TO 280
      WRITE(TERM,150) LINMAX
      GO TO 320
C
C   READ ERROR MODE
C
  280 CONTINUE
      WRITE(TERM,290)
  290 FORMAT('$ERROR MODE=? (M=MANUAL,T=TAPE,N=NO ERRORS)')
      READ(TERM,110,ERR=280) ERRMOD
      IF(ERRMOD.EQ.MM) GO TO 300
      IF(ERRMOD.EQ.TT) GO TO 315
      IF(ERRMOD.NE.NN) GO TO 280
      GO TO 350
C
C   READ ERROR LOCATIONS
C
  300 CONTINUE
      ERRLIM=1
  305 READ(TERM,140) ERRORS(ERRLIM)
      IF(ERRORS(ERRLIM).EQ.9999) GO TO 310
      ERRLIM=ERRLIM+1
      GO TO 305
  310 CONTINUE
      ERRLIM=ERRLIM-1
      GO TO 350
C
C   READ ERROR TAPE FILE AND OPEN
C
  315 CONTINUE
C
      ERRLIM=1
      READ(ERFIL,318,END=317) ERRORS(ERRLIM)
      ERRLIM=ERRLIM+1
  316 READ(ERFIL,318,END=317) ERRORS(ERRLIM)
  318 FORMAT(I16)
      ERRORS(ERRLIM)=ERRORS(ERRLIM)+ERRORS(ERRLIM-1)
      ERRLIM=ERRLIM+1
      GO TO 316
  317 ERRLIM=ERRLIM-1
C
  350 CONTINUE
C
  360 CONTINUE
C   WRITE INPUT PARAMETERS
C
      WRITE(LPFIL,400) PELMAX,VRES,K,EPHASE,CMPMAX,LINMAX
  400 FORMAT('1INPUT PARAMETERS:'/
     *        '0MAXIMUM NUMBER OF PELS PER LINE=',I6/
     *        '0VERTICAL SAMPLING: N=',I4/
     *        '0PARAMETER K =',I4/
     *        '0ERROR PATTERN PHASE =',I4/
     *        '0MINIMUM COMPRESSED LINE LENGTH =',I4,' BITS'/
     *        '0NUMBER OF SCAN LINES TO BE PROCESSED =',I6)
      IF(ERRMOD.EQ.NN) WRITE(LPFIL,410)
  410 FORMAT ('0NO ERRORS INSERTED')
      IF(ERRMOD.EQ.MM) WRITE(TERM,140) (ERRORS(I),I=1,ERRLIM)
```

```
      IF(ERRMOD.EQ.TT) WRITE(TERM,420) ERRLIM
  420 FORMAT(I12,' ERRORS OBTAINED FROM ERROR TAPE')
C******************** BEGIN PROGRAM **************************************
C
C    INITIALIZE
C
      TCDEL=0
      TCDATA=0
      ERRPNT=1
      ERRCNT=0
      INLNCT=0
      ERROFF=EPHASE*1024
      CDELCT=32
      OTELP=1
      CDELP=32+1
      CONSEC=1
      INREF=1
      INCOD=2
      OTREF=1
      OTCOD=2
      STFBIT=0
C
      DO 800 I=1,240
      STFBUF(I)=0
      CDBUF(I)=0
  800 CONTINUE
      DO 850 I=1,60
      OTBUF(I,OTREF)=0
      OTBUF(I,OTCOD)=0
      PELBUF(I,INREF)=0
      PELBUF(I,INCOD)=0
  850 CONTINUE
      SEARCH=.TRUE.
      SYNC=.FALSE.
      WRITE=.FALSE.
C
C    SEARCH MODE: LOOK FOR EOL1 BIT-BY-BIT
C
  900 CONTINUE
      CALL GETLI(13,MODE,LBITS,L)
      GO TO (910,930,930,920),MODE
      STOP 900
  910 CONTINUE
C
C    EOL NOT FOUND; ADVANCE POINTER AND TRY AGAIN
C
      CDELP=CDELP+1
      GO TO 900
  920 CONTINUE
      STOP 920
  930 CONTINUE
C
C    EOL FOUND
C
      SEARCH=.FALSE.
      CDELP=CDELP+L
      IF(WRITE) GO TO 935
      WRITE=.TRUE.
      GO TO 960
  935 CONTINUE
C
C    SET OUTPUT DECODE LINE TO 0 AND WRITE OUT
      DO 950 I=1,60
      OTBUF(I,OTCOD)=0
  950 CONTINUE
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
  960 CONTINUE
      IF(MODE-2)965,1000,900
  965 STOP 965
 1000 CONTINUE
C
C    PERFORM ONE-DIMENSIONAL DECODE OF A COMPLETE LINE
C    FIRST,SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 1010 I=1,60
      OTBUF(I,OTCOD)=0
 1010 CONTINUE
C
      INDEX=3
      COLOR=1
```

```
      OTELP=1
      BICNT=0
C
 1020 CONTINUE
      L=0
      CALL ONEIBM(INDEX,COLOR,STATUS,L)
      GO TO (1030,1070,1070,1035,1040),STATUS
C              1    2    3    4    5
      STOP 1000
C
C     RUN ADDED; CHECK LENGTH OF OUTPUT LINE
C
 1030 CONTINUE
      ONE=.TRUE.
      IF(OTELP-1-PELMAX) 1031,1032,1050
 1031 CONTINUE
      IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
      INDEX=3
      GO TO 1020
 3000 CONTINUE
C
C     PERFORM TWO-DIMENSIONAL DECODE
C
C
C     FIRST, SET OUTPUT BUFFER TO WHITE
C     (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 3010 I=1,60
      OTBUF(I,OTCOD)=0
 3010 CONTINUE
C
      INDEX=1
      COLOR=1
      OTELP=1
      RLFLAG=.FALSE.
      EPFLAG=.FALSE.
C
 3020 CONTINUE
      CALL TWOIBM(INDEX,COLOR,STATUS,L)
      GO TO (3030,1070,1070,1035,1040),STATUS
C              1    2    3    4    5
      STOP 3000
C
C     RUN ADDED; LOOK FOR NEXT RUN
C
 3030 CONTINUE
      ONE=.FALSE.
      IF(OTELP-1-PELMAX) 3031,3032,1050
 3031 CONTINUE
      IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
      INDEX=1
      GO TO 3020
C
C     LINELENGTH = PELMAX; CHECK RL PREFIX
C
 3032 CONTINUE
      RLOREZ=2
      IF(RLFLAG) RLOREZ=1
      CALL GETLI(RLOREZ,MODE,LBITS,L)
      GO TO (3034,1050,1050,1050),MODE
 3034 CONTINUE
      CDELP=CDELP+L
      IF(LBITS.NE.1)GO TO 1070
C
C
C     LINE LENGTH=PELMAX; CHECK FOR FILL AND LOOK FOR EOL
C
 1032 CONTINUE
      ZERO=-1
 1033 CONTINUE
      ZERO=ZERO+1
      CALL GETLI(1,MODE,LBITS,L)
C
      GO TO (1034,1050,1050,1050),MODE
C
C     CHECK FOR FILL
C
 1034 CONTINUE
C
      CDELP=CDELP+L
      IF(LBITS.EQ.0) GO TO 1033
      IF(ZERO.LE.10) GO TO 1070
```

```
C
C     EOL FOUND; CHECK TYPE
C
      CALL GETLI(1,MODE,LBITS,L)
      IF(LBITS.EQ.0) MODE=2
      IF(LBITS.EQ.1) MODE=3
      GO TO (1070,1060,1060,1080),MODE
C
C     PREMATURE EOL DETECTED
C
C
C     EOL1 DETECTED
C
 1035 CONTINUE
      CDELP=CDELP+L
      STATUS=4
      IF(OTELP.LE.5) CONSEC=CONSEC+1
      IF(CONSEC-2)1080,1000,2000
C
C     EOL2 DETECTED
C
 1040 CONTINUE
      CDELP=CDELP+L
      STATUS=5
C
      GO TO 1080
C
C     PROBLEMS,PROBLEMS
C
 1050 STOP 1050
C
C     LINE LENGTH CORRECT, EOL DETECTED PROPERLY; WRITE OUTPUT LINE
C
 1060 CONTINUE
      CDELP=CDELP+L
      WRITE(OTFIL)OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      CTLNNO=LNNOBF
      CONSEC=1
      IF(ONE) SYNC=.TRUE.
      TEMP=OTREF
      OTREF=OTCOD
      OTCOD=TEMP
      IF(MODE.EQ.2) GO TO 1000
      GO TO 3000
C
C     LINE TOO LONG OR NO MATCH
C
 1070 CONTINUE
      WRITE=.FALSE.
C
C     LINE SHORT
C
 1080 CONTINUE
      IF(.NOT.SYNC) GO TO 1090
C
C     WRITE LAST GOOD LINE
C
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTREF),I=1,60)
      SYNC=.FALSE.
      GO TO 1110
 1090 CONTINUE
C
C     WRITE A WHITE LINE
C
      DO 1100 I=1, 60
 1100 OTBUF(I,OTCOD)=0
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
 1110 CTLNNO=LNNOBF
      IF(STATUS.EQ.4) GO TO 1000
      SEARCH=.TRUE.
      GO TO 900
C
C     END OF MESSAGE
C
 2000 CONTINUE
      WRITE(LPFIL,2010) CONSEC
 2010 FORMAT('0END OF MESSAGE DETECTED (',I2,' EOL''S)')
C
C     REPORT COMPRESSION FACTOR, ERROR SENSITIVITY FACTOR,BIT ERROR RATE
C
      ERRATE=FLOAT(ERRCNT)/FLOAT(TCDEL)
      WRITE(LPFIL,2020) TCDEL,TCDATA,STFBIT,INLNCT,ERRATE
```

```
 2020 FORMAT('0TOTAL NUMBER OF CODED BITS = ',I8/
     *       '0TOTAL NUMBER OF CODED DATA BITS = ',I8/
     *       '0TOTAL NUMBER OF 2-DIM LINES = ',I8/
     *       '0TOTAL NUMBER OF INPUT LINES PROCESSED = ',I8/
     *       '0BIT ERROR RATE = ',G14.6)
C
      CALL STATS(STAT,INLNCT,DIAG)
      CF3=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDEL)
      CF4=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDATA)
C
      WRITE(LPFIL,2030) CF3,CF4
 2030 FORMAT('0COMPRESSION FACTOR FOR G3 MACHINE (CF3) =',F8.4/
     *       '0COMPRESSION FACTOR FOR G4 MACHINE (CF4) =',F8.4)
C
      CALL ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      STOP
      E N D
      SUBROUTINE GETLI(LBITS,MODE,WRD,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *              STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNC2F,BICNT,
     *              INCOD,INREF,OTCOD,CTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,RLFLAG,EPFLAG
C***************************** BEGIN PROGRAM ***************************
C
      MODE=4
C
C   RETRIEVE NEXT BIT FROM CDBUF
C
 100  CONTINUE
C
C   ENCODE A NEW LINE IF NECESSARY
C
      IF(LBITS+CDELP-1.LE.CDELCT) GO TO 200
      IF(CDELCT-CDELP+1) 170,190,180
  170 STOP 170
  180 CONTINUE
      STFBUF(1)=I4B(STFBUF,CDELP,CDELCT-CDELP+1)
  190 CONTINUE
      CDELP=32-(CDELCT-CDELP)
      CALL ENCODI
 200  CONTINUE
      WRD=I4B(STFBUF,CDELP,LBITS)
      L=LBITS
      IF(L.LT.13) GO TO 250
      IF(WRD.EQ.CODERD(3,6)) GO TO 300
      IF(WRD.EQ.CODERD(3,7)) GO TO 400
  250 CONTINUE
      MODE=1
      RETURN
  300 CONTINUE
      MODE=2
      RETURN
  400 CONTINUE
      MODE=3
      RETURN
      END
      SUBROUTINE ENCODI
C
      IMPLICIT INTEGER(A-Z)
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
```

```
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *              STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C************************** FILE DEFINITICNS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNOBF,BICNT,
     *              INCOD,INREF,CTCOD,CTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,RLFLAG,EPFLAG
      LOGICAL RLFE,EPFE
C
C************************************** BEGIN PROGRAM ********************
C
C     INITIALIZE VARIABLES
C
      CDELCT=32
      CDDATA=0
      DO 50 I=2,240
      CDBUF(I)=0
      STFBUF(I)=0
   50 CONTINUE
C
C     READ INPUT PICTURE FILE
C
  100 CONTINUE
      READ(PELFIL,END=120,ERR=500)
     * INLNNO,INELCT,(PELBUF(I,INCOD),I=1,60)
      IF(MOD(INLNNO,100).EQ.0)WRITE(TERM,110) INLNNO
  110 FORMAT(' INPUT LINE NO. =',I6)
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
      IF(INELCT.LT.PELMAX) CALL EXIT
      INLNCT=INLNCT+1
C
C     LOAD OUTPUT LINE NUMBER BUFFER
C
      LNNOBF=INLNNO
      IF(SEARCH)OTLNNO=LNNOBF
C
      IF(INLNNO.LE.LINMAX) GO TO 140
C
C     WRITE SIX EOL1'S
C
  120 CONTINUE
      DO 130 I=1,6
      CALL CODIBM(6,0,0,0,0,CDELCT,CDDATA)
  130 CONTINUE
      DO 135 I=1,6
      STFBUF(I)=CDBUF(I)
  135 CONTINUE
      GO TO 400
C
C     FIRST OF K LINES?
C
  140 CONTINUE
      IF(MOD(INLNCT-1,K).NE.0) GO TO 600
C
C     ONE-DIMENSIONAL CODING
C     WRITE ONE EOL1
C
      CALL CODIBM(6,0,0,0,0,CDELCT,CDDATA)
C
      POLAR=1
C
C     TEST COLOR OF FIRST ELEMENT
C
      IF(I4B(PELBUF(1,INCOD),1,1).EQ.0) GO TO 150
C
C     FIRST ELEMENT BLACK; ENCODE 0-LENGTH WHITE RUN
C
      CALL CODELN(0,1,CDELCT,CDDATA)
      POLAR=2
```

```
C
C     CALCULATE RUN LENGTH AND ENCODE
C
  150 CONTINUE
      RUN=0
      DO 200 I=1,PELMAX
      PEL=I4B(PELBUF(1,INCOD),I,1)+1
      IF(PEL.EQ.POLAR) GO TO 180
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 170
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
  160 FORMAT(4I8)
  170 CONTINUE
      RUN=1
      POLAR=MOD(POLAR+2,2)+1
      GO TO 200
  180 CONTINUE
      RUN=RUN+1
  200 CONTINUE
      CALL CODELN(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 210
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
      GO TO 210
C
C     TWO-DIMENSIONAL CODING
C
  600 CONTINUE
C
C     WRITE ONE EOL2
C
      CALL CODIBM(7,0,0,0,0,CDELCT,CDDATA)
C
C     SET A0 TO LEFT EDGE-1 AND POLARITY=WHITE
C
      A0=0
      POL=0
      LEFT=.TRUE.
      RLFE=.FALSE.
      EPFE=.FALSE.
C
C     DETECT A1
C
  620 CONTINUE
      I=A0+1
      IF(I.GT.PELMAX) GO TO 640
  630 CONTINUE
      PEL=I4B(PELBUF(1,INCOD),I,1)
      IF(PEL.NE.POL) GO TO 640
      I=I+1
      IF(I.LE.PELMAX) GO TO 630
  640 CONTINUE
      A1=I
C
C     DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 665
      PELM1=I4B(PELBUF(1,INREF),A0,1)
      IF(LEFT) PELM1=0
  650 CONTINUE
      PEL=I4B(PELBUF(1,INREF),I,1)
      IF(PEL.NE.PELM1) GO TO 670
  660 CONTINUE
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 650
  665 CONTINUE
      B1=I
      GO TO 730
  670 CONTINUE
      IF(PEL.NE.POL) GO TO 690
      GO TO 660
  690 CONTINUE
      B1=I
C
C     ADJUST LEFT EDGE IF NECESSARY
C
  730 CONTINUE
      IF(.NOT.LEFT) POLAR=I4B(PELBUF(1,INCOD),A0,1)+1
      IF(.NOT.LEFT) GO TO 740
      POLAR=1
      A0=1
```

```
      LEFT=.FALSE.
  740 CONTINUE
C
C   TEST COLOR OF FIRST ELEMENT ON LINE
C
      IF(A1.NE.1) GO TO 750
      IF(I48(PELBUF(1,INCOD),1,1).EQ.1) GO TO 800
C
C   FIRST ELEMENT WHITE
C
  750 CONTINUE
C
C   CALCULATE VERTICAL LENGTH
C
      MAB=IABS(A1-B1)
      IF(MAB-1) 850,850,800
C
C
C   CODE BY HORIZONTAL MODE
C
  800 CONTINUE
C
C   TEST FOR RUNLENGTH REDUCTION
C
      REDUC=0
      IF(B1.GE.A1-1) GO TO 810
      IF(A0.EQ.1) GO TO 810
C
C   CONDITION FOR RUNLENGTH REDUCTION MET
C
      REDUC=MIN0(B1-A0+1,3)
C
C   CHECK RL CORRELATION
C
  810 CONTINUE
      RLOREZ=2
      IF(RLFE) RLOREZ=1
      CALL CODIBM(RLOREZ,POLAR,A0,A1-REDUC,.TRUE.,CDELCT,CDDATA)
      RLFE=.TRUE.
      EPFE=.FALSE.
      A0=A1
      GO TO 960
C
C   CODE BY VERTICAL MODE
C
  850 CONTINUE
      IF(A1-B1) 870,880,900
C
  870 CALL CODIBM(3,0,0,0,0,CDELCT,CDDATA)
      EPFE=.FALSE.
      GO TO 950
  880 CONTINUE
      RLOREZ=1
      IF(RLFE) RLOREZ=2
      CALL CODIBM(RLOREZ,0,0,0,0,CDELCT,CDDATA)
      EPFE=.FALSE.
      GO TO 950
C
  900 CONTINUE
      IF(EPFE) GO TO 920
  910 CALL CODIBM(4,0,0,0,0,CDELCT,CDDATA)
      EPFE=.TRUE.
      GO TO 950
  920 CONTINUE
      IF(I48(CDBUF,CDELCT,1).EQ.1) GO TO 910
      CALL CODIBM(5,0,0,0,0,CDELCT,CDDATA)
      EPFE=.TRUE.
  950 CONTINUE
      RLFE=.FALSE.
      A0=A1
C
C   TEST FOR END OF LINE
C
  960 CONTINUE
      IF(A0.GT.PELMAX) GO TO 205
      POL=I48(PELBUF(1,INCOD),A0,1)
      GO TO 620
  205 CONTINUE
      RLOREZ=2
      IF(RLFE) RLOREZ=1
      CALL CODIBM(RLOREZ,0,0,0,.FALSE.,CDELCT,CDDATA)
  210 CONTINUE
```

```
C
C     SWITCH CODE & REFERENCE LINES
C
      TEMP=INREF
      INREF=INCOD
      INCOD=TEMP
C
C    TRANSFER CDBUF TO STFBUF
C
      CDELW=(CDELCT+32-1)/32
      DO 240 I=2,CDELW
      STFBUF(I)=CDBUF(I)
  240 CONTINUE
C
C    SAVE LINE LENGTH(DATA BITS PLUS EOL)
C
      STAT(INLNCT)=CDDATA+CODERD(1,7)
C
C    CHECK CODED LINE LENGTH
C
      FILL=CMPMAX-(CDELCT-32)
      IF(FILL) 400,400,250
C
C    CODE LINE TOO SHORT; FILL IT TO CMPMAX
  250 CONTINUE
      CDELCT=CDELCT+FILL
C
C    ACCUMULATE STATISTICS AND ERROR CORRUPT
C
  400 CONTINUE
      IF(ERRMOD.EQ.NN) GO TO 390
C
C    ERROR CORRUPT
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C    ERROR IN RANGE OF CODED LINE; CHANGE APPROPRIATE BIT
C
      BIT=I4B(STFBUF,ERRBIT+32,1)
      BIT=MOD(BIT+1,2)
      CALL MI2B(BIT,STFBUF,ERRBIT+32,1)
      ERRCNT=ERRCNT+1
C
C    INCREMENT ERROR LIST POINTER
C
  360 CONTINUE
      ERRPNT=ERRPNT+1
      IF(ERRPNT.LE.ERRLIM) GO TO 350
C
C    ERROR LIST EXHAUSTED
C
      ERRPNT=ERRPNT-1
      WRITE(LPFIL,370) ERRPNT,ERRORS(ERRPNT)
  370 FORMAT('0ERROR LIST EXHAUSTED AT',I10,'TH ERROR:'/
     *       ' LAST ERROR OCCURRED AT',I10,' BITS')
      ERRMOD=NN
C
C    COMPUTE STATISTICS
C
  390 CONTINUE
      TCDEL=TCDEL+CDELCT-32
      TCDATA=TCDATA+CDDATA
      IF(DIAG) WRITE(TERM,160) INLNCT, CDDATA
C
      IF (.NOT.DIAG) GO TO 460
      CDELW=(CDELCT+32-1)/32
      WRITE(LPFIL,450) (CDBUF(I),I=1,CDELW)
      WRITE(LPFIL,450) (STFBUF(I),I=1,CDELW)
  450 FORMAT(6Z12)
  460 CONTINUE
      RETURN
C
  500 CONTINUE
      CALL EXIT
C
      END
      SUBROUTINE CODIBM(MODE,POLAR,A,B,RLF,CDELCT,CDDATA)
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
```

```
     *              STFBUF(240), STAT(3000)
      COMMON/HJFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
      LOGICAL RLF
C
C**************************** BEGIN PRCGRAM ***************************
C
      CALL MI28(CODERD(3,MODE),CDBUF,CDELCT+1,CODEPD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MCDE)
      GC TJ (200,200,100,100,100,800,800) ,MCDE
C
C   MODE      1   2   3   4   5   6   7
C
      STOP 129
C
C   EM OR EP (3,4,5)
C
 100  CONTINUE
      CDDATA=CDDATA+CODERD(1,MODE)
      RETURN
C
C   RL OR EZ (1,2)
C
 200 CONTINUE
      CDDATA=CDDATA+CODERD(1,MODE)
      IF(RLF)CALL CODELN(B-A,POLAR,CDELCT,CDDATA)
      RETURN
C
C   ADD EOL1 OR EOL2 TO LINE (6,7)
C
 800  CONTINUE
      RETURN
      END
      SUBROUTINE ONEIBM(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *              STFBUF(240), STAT(3000)
      COMMON/HJFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE CEFINITICNS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNC,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNCBF,BICNT,
     *              INCOD,INREF,OTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,RLFLAG,EPFLAG
C
C*********************** BEGIN PROGRAM *********************
C
      REDUC=L
C
C   BEGIN DECODE LOOP; RETRIEVE NEXT CODE WCRD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODE(1,INDEX,COLOR)
      CALL GETLI(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z8,I6)
      GO TO (1040,1200,1205,1190), MCDE
      STCP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODE(3,INDEX,COLOR)) GO TO 1100
C
C   NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODE(2,INDEX,COLOR)
      IF(INDEX.GE.93) GO TO 1190
      IF(CODE(1,INDEX,COLOR).EQ.LENBIT) GO TO 1040
C
```

```
C     CODE WORD LONGER; FROM THE TOP
C
         GO TO 1002
C
C     MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C     NOT AN EOL
C
C
C
C     TEST FOR MAKE UP OR TERMINATING CODE
C
      RUNLEN=INDEX-1
      IF(BICNT.LT.OTELP+RUNLEN+REDUC-1) RUNLEN=RUNLEN+REDUC
      IF(INDEX.GE.65) RUNLEN=(INDEX-64)*64
      IF(RUNLEN.EQ.0) GO TO 1160
      IF(COLOR.EQ.1) GO TO 1155
      IF(RUNLEN.LT.0) STOP 1100
C
C     ADD BLACK RUN TO OUTPUT BUFFER
C
      DO 1150 I=1,RUNLEN
      CALL MI2B(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C     ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C OUTPUT LINE LESS THAN OR EQUAL TO MAX SPECIFIED
C
 1160 CONTINUE
      IF(INDEX.LT.65) GO TO 1170
      INDEX=3
      GO TO 1000
C
C     RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1170 CONTINUE
      CHCOL=.TRUE.
      STATUS=1
      RETURN
C
C     RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C     NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C     EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C     EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      SUBROUTINE TWOIBM(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
```

```
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C************************ FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,CTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,B1CNT,
     *            INCOD,INREF,CTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,RLFLAG,EPFLAG
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODERD(1,INDEX)
      CALL GETLI(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z12,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODERD(3,INDEX)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODERD(2,INDEX)
      IF(INDEX.GE.8) GO TO 1190
      IF(CODERD(1,INDEX).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
C
C
C    FIND B1 AND B2
C
      A0=OTELP
      IF(OTELP.EQ.1) A0=0
      POL=COLOR-1
C
C    DETECT B1
C
      I=A0+1
      IF(I.GT.PELMAX) GO TO 65
      PELM1=0
      IF(A0.EQ.0) GO TO 50
      PELM1=I48(OTBUF(1,OTREF),A0,1)
   50 CONTINUE
      PEL=I48(OTBUF(1,OTREF),I,1)
      IF(PEL.NE.PELM1) GO TO 70
   60 CONTINUE
      PELM1=PEL
      I=I+1
      IF(I.LE.PELMAX) GO TO 50
   65 CONTINUE
      B1=I
      GO TO 92
   70 CONTINUE
      IF(PEL.NE.POL) GO TO 90
      GO TO 60
   90 CONTINUE
      B1=I
   92 CONTINUE
      GO TO (100,110,600,400,410),INDEX
      STOP 100
```

```
C
C     RL OR EZ; DECIDE WHICH
C
 100  CONTINUE
      IF(RLFLAG) GO TO 200
      GO TO 300
  110 CONTINUE
      IF(RLFLAG) GO TO 300
      GO TO 200
C
C     HORIZONTAL MODE RL
C
 200  CONTINUE
      RLFLAG=.TRUE.
      EPFLAG=.FALSE.
      ENTRY=3
C
C     SET UP TO ADD BACK RUNLENGTH REDUCTION IF APPLICABLE
C
      L=0
      B1CNT=B1
      IF(A0.EQ.0) GO TO 205
      L=MIN0(B1-A0+1,3)
  205 CONTINUE
      CALL ONEIBM(ENTRY,COLOR,STATE,L)
      GO TO (210,1160,1190,1200,1205),STATE
 210  CONTINUE
      CHCOL=.TRUE.
      GO TO 1160
C
C     VERTICAL MODE A1B1=0 EZ
C
 300  CONTINUE
      RUNLEN=B1-OTELP
      RLFLAG=.FALSE.
      EPFLAG=.FALSE.
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     EP 000 OR 0001
C
  400 CONTINUE
      IF(.NOT.EPFLAG) GO TO 500
C
C     EP FOLLOWING EP(000); NEXT BIT MUST BE A '1'
C     OR AN ERROR CONDITION HAS BEEN DETECTED
C
      CALL GET_I(1,MODE,LBITS,L)
      GO TO (410,405,405,405),MODE
  405 STOP 405
  410 CONTINUE
      CDELP=CDELP+L
      IF(LBITS) 405,420,480
  420 CONTINUE
C
C     FOUR CONSECUTIVE ZEROES DETECTED
C
      CDELP=CDELP-4
      GO TO 1190
C
C     VERTICAL MODE RIGHT EP A1B1=1
C
  480 CONTINUE
      EPFLAG=.FALSE.
      GO TO 510
  500 CONTINUE
      EPFLAG=.TRUE.
  510 CONTINUE
      RUNLEN=B1-OTELP+1
      RLFLAG=.FALSE.
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
C     VERTICAL MODE LEFT EM A1B1=1
C
 600  CONTINUE
      EPFLAG=.FALSE.
      RLFLAG=.FALSE.
      RUNLEN=B1-OTELP-1
      CHCOL=.TRUE.
      GO TO (1155,1145),COLOR
C
```

```
C     ADD BLACK RUN TO OUTPUT BUFFER
C
 1145 CONTINUE
      IF(RUNLEN)1190,1160,1147
 1147 CONTINUE
      DO 1150 I=1,RUNLEN
      CALL MI23(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C     ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      IF(RUNLEN.LT.0) GO TO 1190
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C     RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1160 CONTINUE
      STATUS=1
      RETURN
C
C     RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C     NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C     EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C     EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      END
      BLOCK DATA
C
      IMPLICIT INTEGER(A-Z)
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *             STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,9)
      COMMON/ERAY/ERRORS(2500)
C********************* LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *             CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *             ERRCNT,INLNCT,CONSEC,LNNCBF,BICNT,
     *             INCOD,INREF,OTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *RLFLAG,EPFLAG
      LOGICAL SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL RLFLAG,EPFLAG
C
      DATA TERM,LPFIL,PELFIL,OTFIL,ERFIL/5,6,1,2,3/
      DATA DD,II,MM,TT,NN,YY/'D','I','M','T','N','Y'/
      DATA PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX/1728,2,0,96,'T',3000/
      DATA K/2/
      DATA DIAG/.FALSE./
C
      DATA CODE(1,  1,1),CODE(2,  1,1),CODE(3,  1,1)/ 8, 70,Z0035/
```

```
DATA CODE(1,  2,1),CODE(2,  2,1),CODE(3,  2,1)/ 6, 90,Z0007/
DATA CODE(1,  3,1),CODE(2,  3,1),CODE(3,  3,1)/ 4,  4,Z0007/
DATA CODE(1,  4,1),CODE(2,  4,1),CODE(3,  4,1)/ 4,  5,Z0008/
DATA CODE(1,  5,1),CODE(2,  5,1),CODE(3,  5,1)/ 4,  6,Z000B/
DATA CODE(1,  6,1),CODE(2,  6,1),CODE(3,  6,1)/ 4,  7,Z000C/
DATA CODE(1,  7,1),CODE(2,  7,1),CODE(3,  7,1)/ 4,  8,Z000E/
DATA CODE(1,  8,1),CODE(2,  8,1),CODE(3,  8,1)/ 4,  9,Z000F/
DATA CODE(1,  9,1),CODE(2,  9,1),CODE(3,  9,1)/ 5, 10,Z0013/
DATA CODE(1, 10,1),CODE(2, 10,1),CODE(3, 10,1)/ 5, 11,Z0014/
DATA CODE(1, 11,1),CODE(2, 11,1),CODE(3, 11,1)/ 5, 12,Z0007/
DATA CODE(1, 12,1),CODE(2, 12,1),CODE(3, 12,1)/ 5, 65,Z0008/
DATA CODE(1, 13,1),CODE(2, 13,1),CODE(3, 13,1)/ 6, 14,Z0008/
DATA CODE(1, 14,1),CODE(2, 14,1),CODE(3, 14,1)/ 6, 15,Z0003/
DATA CODE(1, 15,1),CODE(2, 15,1),CODE(3, 15,1)/ 6, 16,Z0034/
DATA CODE(1, 16,1),CODE(2, 16,1),CODE(3, 16,1)/ 6, 17,Z0035/
DATA CODE(1, 17,1),CODE(2, 17,1),CODE(3, 17,1)/ 6, 18,Z002A/
DATA CODE(1, 18,1),CODE(2, 18,1),CODE(3, 18,1)/ 6, 19,Z002B/
DATA CODE(1, 19,1),CODE(2, 19,1),CODE(3, 19,1)/ 7, 20,Z0027/
DATA CODE(1, 20,1),CODE(2, 20,1),CODE(3, 20,1)/ 7, 21,Z000C/
DATA CODE(1, 21,1),CODE(2, 21,1),CODE(3, 21,1)/ 7, 22,Z0008/
DATA CODE(1, 22,1),CODE(2, 22,1),CODE(3, 22,1)/ 7, 23,Z0017/
DATA CODE(1, 23,1),CODE(2, 23,1),CODE(3, 23,1)/ 7, 24,Z0003/
DATA CODE(1, 24,1),CODE(2, 24,1),CODE(3, 24,1)/ 7, 25,Z0004/
DATA CODE(1, 25,1),CODE(2, 25,1),CODE(3, 25,1)/ 7, 26,Z0028/
DATA CODE(1, 26,1),CODE(2, 26,1),CODE(3, 26,1)/ 7, 27,Z002B/
DATA CODE(1, 27,1),CODE(2, 27,1),CODE(3, 27,1)/ 7, 28,Z0013/
DATA CODE(1, 28,1),CODE(2, 28,1),CODE(3, 28,1)/ 7, 29,Z0024/
DATA CODE(1, 29,1),CODE(2, 29,1),CODE(3, 29,1)/ 7, 68,Z0018/
DATA CODE(1, 30,1),CODE(2, 30,1),CODE(3, 30,1)/ 8, 31,Z0002/
DATA CODE(1, 31,1),CODE(2, 31,1),CODE(3, 31,1)/ 8, 32,Z0003/
DATA CODE(1, 32,1),CODE(2, 32,1),CODE(3, 32,1)/ 8, 33,Z001A/
DATA CODE(1, 33,1),CODE(2, 33,1),CODE(3, 33,1)/ 8, 34,Z001B/
DATA CODE(1, 34,1),CODE(2, 34,1),CODE(3, 34,1)/ 8, 35,Z0012/
DATA CODE(1, 35,1),CODE(2, 35,1),CODE(3, 35,1)/ 8, 36,Z0013/
DATA CODE(1, 36,1),CODE(2, 36,1),CODE(3, 36,1)/ 8, 37,Z0014/
DATA CODE(1, 37,1),CODE(2, 37,1),CODE(3, 37,1)/ 8, 38,Z0015/
DATA CODE(1, 38,1),CODE(2, 38,1),CODE(3, 38,1)/ 8, 39,Z0016/
DATA CODE(1, 39,1),CODE(2, 39,1),CODE(3, 39,1)/ 8, 40,Z0017/
DATA CODE(1, 40,1),CODE(2, 40,1),CODE(3, 40,1)/ 8, 41,Z0028/
DATA CODE(1, 41,1),CODE(2, 41,1),CODE(3, 41,1)/ 8, 42,Z0029/
DATA CODE(1, 42,1),CODE(2, 42,1),CODE(3, 42,1)/ 8, 43,Z002A/
DATA CODE(1, 43,1),CODE(2, 43,1),CODE(3, 43,1)/ 8, 44,Z002B/
DATA CODE(1, 44,1),CODE(2, 44,1),CODE(3, 44,1)/ 8, 45,Z002C/
DATA CODE(1, 45,1),CODE(2, 45,1),CODE(3, 45,1)/ 8, 46,Z002D/
DATA CODE(1, 46,1),CODE(2, 46,1),CODE(3, 46,1)/ 8, 47,Z0004/
DATA CODE(1, 47,1),CODE(2, 47,1),CODE(3, 47,1)/ 8, 48,Z0005/
DATA CODE(1, 48,1),CODE(2, 48,1),CODE(3, 48,1)/ 8, 49,Z000A/
DATA CODE(1, 49,1),CODE(2, 49,1),CODE(3, 49,1)/ 8, 50,Z000B/
DATA CODE(1, 50,1),CODE(2, 50,1),CODE(3, 50,1)/ 8, 51,Z0052/
DATA CODE(1, 51,1),CODE(2, 51,1),CODE(3, 51,1)/ 8, 52,Z0053/
DATA CODE(1, 52,1),CODE(2, 52,1),CODE(3, 52,1)/ 8, 53,Z0054/
DATA CODE(1, 53,1),CODE(2, 53,1),CODE(3, 53,1)/ 8, 54,Z0055/
DATA CODE(1, 54,1),CODE(2, 54,1),CODE(3, 54,1)/ 8, 55,Z0024/
DATA CODE(1, 55,1),CODE(2, 55,1),CODE(3, 55,1)/ 8, 56,Z0025/
DATA CODE(1, 56,1),CODE(2, 56,1),CODE(3, 56,1)/ 8, 57,Z0058/
DATA CODE(1, 57,1),CODE(2, 57,1),CODE(3, 57,1)/ 8, 58,Z0059/
DATA CODE(1, 58,1),CODE(2, 58,1),CODE(3, 58,1)/ 8, 59,Z005A/
DATA CODE(1, 59,1),CODE(2, 59,1),CODE(3, 59,1)/ 8, 60,Z005B/
DATA CODE(1, 60,1),CODE(2, 60,1),CODE(3, 60,1)/ 8, 61,Z004A/
DATA CODE(1, 61,1),CODE(2, 61,1),CODE(3, 61,1)/ 8, 62,Z004B/
DATA CODE(1, 62,1),CODE(2, 62,1),CODE(3, 62,1)/ 8, 63,Z0032/
DATA CODE(1, 63,1),CODE(2, 63,1),CODE(3, 63,1)/ 8, 64,Z0033/
DATA CODE(1, 64,1),CODE(2, 64,1),CODE(3, 64,1)/ 8, 69,Z0034/
DATA CODE(1, 65,1),CODE(2, 65,1),CODE(3, 65,1)/ 5, 66,Z001B/
DATA CODE(1, 66,1),CODE(2, 66,1),CODE(3, 66,1)/ 5, 67,Z0012/
DATA CODE(1, 67,1),CODE(2, 67,1),CODE(3, 67,1)/ 6,  2,Z0017/
DATA CODE(1, 68,1),CODE(2, 68,1),CODE(3, 68,1)/ 7, 30,Z0037/
DATA CODE(1, 69,1),CODE(2, 69,1),CODE(3, 69,1)/ 8,  1,Z0036/
DATA CODE(1, 70,1),CODE(2, 70,1),CODE(3, 70,1)/ 8, 71,Z0037/
DATA CODE(1, 71,1),CODE(2, 71,1),CODE(3, 71,1)/ 8, 72,Z0064/
DATA CODE(1, 72,1),CODE(2, 72,1),CODE(3, 72,1)/ 9, 73,Z0065/
DATA CODE(1, 73,1),CODE(2, 73,1),CODE(3, 73,1)/ 8, 74,Z0068/
DATA CODE(1, 74,1),CODE(2, 74,1),CODE(3, 74,1)/ 8, 75,Z0067/
DATA CODE(1, 75,1),CODE(2, 75,1),CODE(3, 75,1)/ 9, 76,Z00CC/
DATA CODE(1, 76,1),CODE(2, 76,1),CODE(3, 76,1)/ 9, 77,Z00CD/
DATA CODE(1, 77,1),CODE(2, 77,1),CODE(3, 77,1)/ 9, 78,Z00D2/
DATA CODE(1, 78,1),CODE(2, 78,1),CODE(3, 78,1)/ 9, 79,Z00D3/
DATA CODE(1, 79,1),CODE(2, 79,1),CODE(3, 79,1)/ 9, 80,Z00D4/
DATA CODE(1, 80,1),CODE(2, 80,1),CODE(3, 80,1)/ 9, 81,Z00D5/
DATA CODE(1, 81,1),CODE(2, 81,1),CODE(3, 81,1)/ 9, 82,Z00D6/
DATA CODE(1, 82,1),CODE(2, 82,1),CODE(3, 82,1)/ 9, 83,Z00D7/
DATA CODE(1, 83,1),CODE(2, 83,1),CODE(3, 83,1)/ 9, 84,Z00D8/
```

```
DATA CODE(1, 84,1),CODE(2, 84,1),CODE(3, 84,1)/ 9, 85,Z0009/
DATA CODE(1, 85,1),CODE(2, 85,1),CODE(3, 85,1)/ 9, 86,Z00D4/
DATA CODE(1, 86,1),CODE(2, 86,1),CODE(3, 86,1)/ 9, 87,Z00D6/
DATA CODE(1, 87,1),CODE(2, 87,1),CODE(3, 87,1)/ 9, 88,Z0098/
DATA CODE(1, 88,1),CODE(2, 88,1),CODE(3, 88,1)/ 9, 89,Z0099/
DATA CODE(1, 89,1),CODE(2, 89,1),CODE(3, 89,1)/ 9, 91,Z009A/
DATA CODE(1, 90,1),CODE(2, 90,1),CODE(3, 90,1)/ 6, 13,Z0018/
DATA CODE(1, 91,1),CODE(2, 91,1),CODE(3, 91,1)/ 9, 92,Z009B/
DATA CODE(1, 92,1),CODE(2, 92,1),CODE(3, 92,1)/13, 93,Z0002/
DATA CODE(1,  1,2),CODE(2,  1,2),CODE(3,  1,2)/10, 65,Z0037/
DATA CODE(1,  2,2),CODE(2,  2,2),CODE(3,  2,2)/ 3,  6,Z0002/
DATA CODE(1,  3,2),CODE(2,  3,2),CODE(3,  3,2)/ 2,  4,Z0003/
DATA CODE(1,  4,2),CODE(2,  4,2),CODE(3,  4,2)/ 2,  5,Z0002/
DATA CODE(1,  5,2),CODE(2,  5,2),CODE(3,  5,2)/ 3,  2,Z0003/
DATA CODE(1,  6,2),CODE(2,  6,2),CODE(3,  6,2)/ 4,  7,Z0003/
DATA CODE(1,  7,2),CODE(2,  7,2),CODE(3,  7,2)/ 4,  8,Z0002/
DATA CODE(1,  8,2),CODE(2,  8,2),CODE(3,  8,2)/ 5,  9,Z0003/
DATA CODE(1,  9,2),CODE(2,  9,2),CODE(3,  9,2)/ 6, 10,Z0005/
DATA CODE(1, 10,2),CODE(2, 10,2),CODE(3, 10,2)/ 6, 11,Z0004/
DATA CODE(1, 11,2),CODE(2, 11,2),CODE(3, 11,2)/ 7, 12,Z0004/
DATA CODE(1, 12,2),CODE(2, 12,2),CODE(3, 12,2)/ 7, 13,Z0005/
DATA CODE(1, 13,2),CODE(2, 13,2),CODE(3, 13,2)/ 7, 14,Z0007/
DATA CODE(1, 14,2),CODE(2, 14,2),CODE(3, 14,2)/ 8, 15,Z0004/
DATA CODE(1, 15,2),CODE(2, 15,2),CODE(3, 15,2)/ 8, 16,Z0007/
DATA CODE(1, 16,2),CODE(2, 16,2),CODE(3, 16,2)/ 9, 17,Z0018/
DATA CODE(1, 17,2),CODE(2, 17,2),CODE(3, 17,2)/10, 18,Z0017/
DATA CODE(1, 18,2),CODE(2, 18,2),CODE(3, 18,2)/10, 19,Z0018/
DATA CODE(1, 19,2),CODE(2, 19,2),CODE(3, 19,2)/10,  1,Z0008/
DATA CODE(1, 20,2),CODE(2, 20,2),CODE(3, 20,2)/11, 21,Z0067/
DATA CODE(1, 21,2),CODE(2, 21,2),CODE(3, 21,2)/11, 22,Z0068/
DATA CODE(1, 22,2),CODE(2, 22,2),CODE(3, 22,2)/11, 23,Z006C/
DATA CODE(1, 23,2),CODE(2, 23,2),CODE(3, 23,2)/11, 24,Z0037/
DATA CODE(1, 24,2),CODE(2, 24,2),CODE(3, 24,2)/11, 25,Z0028/
DATA CODE(1, 25,2),CODE(2, 25,2),CODE(3, 25,2)/11, 26,Z0017/
DATA CODE(1, 26,2),CODE(2, 26,2),CODE(3, 26,2)/11, 27,Z0018/
DATA CODE(1, 27,2),CODE(2, 27,2),CODE(3, 27,2)/12, 28,Z00CA/
DATA CODE(1, 28,2),CODE(2, 28,2),CODE(3, 28,2)/12, 29,Z00CB/
DATA CODE(1, 29,2),CODE(2, 29,2),CODE(3, 29,2)/12, 30,Z00CC/
DATA CODE(1, 30,2),CODE(2, 30,2),CODE(3, 30,2)/12, 31,Z00CD/
DATA CODE(1, 31,2),CODE(2, 31,2),CODE(3, 31,2)/12, 32,Z0068/
DATA CODE(1, 32,2),CODE(2, 32,2),CODE(3, 32,2)/12, 33,Z0069/
DATA CODE(1, 33,2),CODE(2, 33,2),CODE(3, 33,2)/12, 34,Z006A/
DATA CODE(1, 34,2),CODE(2, 34,2),CODE(3, 34,2)/12, 35,Z006B/
DATA CODE(1, 35,2),CODE(2, 35,2),CODE(3, 35,2)/12, 36,Z00D2/
DATA CODE(1, 36,2),CODE(2, 36,2),CODE(3, 36,2)/12, 37,Z00D3/
DATA CODE(1, 37,2),CODE(2, 37,2),CODE(3, 37,2)/12, 38,Z00D4/
DATA CODE(1, 38,2),CODE(2, 38,2),CODE(3, 38,2)/12, 39,Z00D5/
DATA CODE(1, 39,2),CODE(2, 39,2),CODE(3, 39,2)/12, 40,Z00D6/
DATA CODE(1, 40,2),CODE(2, 40,2),CODE(3, 40,2)/12, 41,Z00D7/
DATA CODE(1, 41,2),CODE(2, 41,2),CODE(3, 41,2)/12, 42,Z006C/
DATA CODE(1, 42,2),CODE(2, 42,2),CODE(3, 42,2)/12, 43,Z006D/
DATA CODE(1, 43,2),CODE(2, 43,2),CODE(3, 43,2)/12, 44,Z00DA/
DATA CODE(1, 44,2),CODE(2, 44,2),CODE(3, 44,2)/12, 45,Z00DB/
DATA CODE(1, 45,2),CODE(2, 45,2),CODE(3, 45,2)/12, 46,Z0054/
DATA CODE(1, 46,2),CODE(2, 46,2),CODE(3, 46,2)/12, 47,Z0055/
DATA CODE(1, 47,2),CODE(2, 47,2),CODE(3, 47,2)/12, 48,Z0056/
DATA CODE(1, 48,2),CODE(2, 48,2),CODE(3, 48,2)/12, 49,Z0057/
DATA CODE(1, 49,2),CODE(2, 49,2),CODE(3, 49,2)/12, 50,Z0064/
DATA CODE(1, 50,2),CODE(2, 50,2),CODE(3, 50,2)/12, 51,Z0065/
DATA CODE(1, 51,2),CODE(2, 51,2),CODE(3, 51,2)/12, 52,Z0052/
DATA CODE(1, 52,2),CODE(2, 52,2),CODE(3, 52,2)/12, 53,Z0053/
DATA CODE(1, 53,2),CODE(2, 53,2),CODE(3, 53,2)/12, 54,Z0024/
DATA CODE(1, 54,2),CODE(2, 54,2),CODE(3, 54,2)/12, 55,Z0037/
DATA CODE(1, 55,2),CODE(2, 55,2),CODE(3, 55,2)/12, 56,Z0038/
DATA CODE(1, 56,2),CODE(2, 56,2),CODE(3, 56,2)/12, 57,Z0027/
DATA CODE(1, 57,2),CODE(2, 57,2),CODE(3, 57,2)/12, 58,Z0028/
DATA CODE(1, 58,2),CODE(2, 58,2),CODE(3, 58,2)/12, 59,Z0058/
DATA CODE(1, 59,2),CODE(2, 59,2),CODE(3, 59,2)/12, 60,Z0059/
DATA CODE(1, 60,2),CODE(2, 60,2),CODE(3, 60,2)/12, 61,Z002B/
DATA CODE(1, 61,2),CODE(2, 61,2),CODE(3, 61,2)/12, 62,Z002C/
DATA CODE(1, 62,2),CODE(2, 62,2),CODE(3, 62,2)/12, 63,Z005A/
DATA CODE(1, 63,2),CODE(2, 63,2),CODE(3, 63,2)/12, 64,Z0066/
DATA CODE(1, 64,2),CODE(2, 64,2),CODE(3, 64,2)/12, 66,Z0067/
DATA CODE(1, 65,2),CODE(2, 65,2),CODE(3, 65,2)/10, 20,Z000F/
DATA CODE(1, 66,2),CODE(2, 66,2),CODE(3, 66,2)/12, 67,Z00C8/
DATA CODE(1, 67,2),CODE(2, 67,2),CODE(3, 67,2)/12, 68,Z00C9/
DATA CODE(1, 68,2),CODE(2, 68,2),CODE(3, 68,2)/12, 69,Z005B/
DATA CODE(1, 69,2),CODE(2, 69,2),CODE(3, 69,2)/12, 70,Z0033/
DATA CODE(1, 70,2),CODE(2, 70,2),CODE(3, 70,2)/12, 71,Z0034/
DATA CODE(1, 71,2),CODE(2, 71,2),CODE(3, 71,2)/12, 72,Z0035/
DATA CODE(1, 72,2),CODE(2, 72,2),CODE(3, 72,2)/13, 73,Z006C/
DATA CODE(1, 73,2),CODE(2, 73,2),CODE(3, 73,2)/13, 74,Z006D/
```

```
      DATA CODE(1, 74,2),CODE(2, 74,2),CODE(3, 74,2)/13, 75,Z004A/
      DATA CODE(1, 75,2),CODE(2, 75,2),CODE(3, 75,2)/13, 76,Z004B/
      DATA CODE(1, 76,2),CODE(2, 76,2),CODE(3, 76,2)/13, 77,Z004C/
      DATA CODE(1, 77,2),CODE(2, 77,2),CODE(3, 77,2)/13, 78,Z004D/
      DATA CODE(1, 78,2),CODE(2, 78,2),CODE(3, 78,2)/13, 79,Z0072/
      DATA CODE(1, 79,2),CODE(2, 79,2),CODE(3, 79,2)/13, 80,Z0073/
      DATA CODE(1, 80,2),CODE(2, 80,2),CODE(3, 80,2)/13, 81,Z0074/
      DATA CODE(1, 81,2),CODE(2, 81,2),CODE(3, 81,2)/13, 82,Z0075/
      DATA CODE(1, 82,2),CODE(2, 82,2),CODE(3, 82,2)/13, 83,Z0076/
      DATA CODE(1, 83,2),CODE(2, 83,2),CODE(3, 83,2)/13, 84,Z0077/
      DATA CODE(1, 84,2),CODE(2, 84,2),CODE(3, 84,2)/13, 85,Z0052/
      DATA CODE(1, 85,2),CODE(2, 85,2),CODE(3, 85,2)/13, 86,Z0053/
      DATA CODE(1, 86,2),CODE(2, 86,2),CODE(3, 86,2)/13, 87,Z0054/
      DATA CODE(1, 87,2),CODE(2, 87,2),CODE(3, 87,2)/13, 88,Z0055/
      DATA CODE(1, 88,2),CODE(2, 88,2),CODE(3, 88,2)/13, 89,Z005A/
      DATA CODE(1, 89,2),CODE(2, 89,2),CODE(3, 89,2)/13, 90,Z005B/
      DATA CODE(1, 90,2),CODE(2, 90,2),CODE(3, 90,2)/13, 91,Z0064/
      DATA CODE(1, 91,2),CODE(2, 91,2),CODE(3, 91,2)/13, 92,Z0065/
      DATA CODE(1, 92,2),CODE(2, 92,2),CODE(3, 92,2)/13, 93,Z0003/
      DATA CODERD(1,1),CODERD(2,1),CODERD(3,1)/        1,2,Z1/
      DATA CODERD(1,2),CODERD(2,2),CODERD(3,2)/        2,3,Z1/
      DATA CODERD(1,3),CODERD(2,3),CODERD(3,3)/        3,4,Z1/
      DATA CODERD(1,4),CODERD(2,4),CODERD(3,4)/        3,5,Z0/
      DATA CODERD(1,5),CODERD(2,5),CODERD(3,5)/        4,6,Z1/
      DATA CODERD(1,6),CODERD(2,6),CODERD(3,6)/        13,7,Z2/
      DATA CODERD(1,7),CODERD(2,7),CODERD(3,7)/        13,8,Z3/
C
      E N D
0          END OF DCEC UPRINT PROGRAM          LINES PRINTED=  1421
```

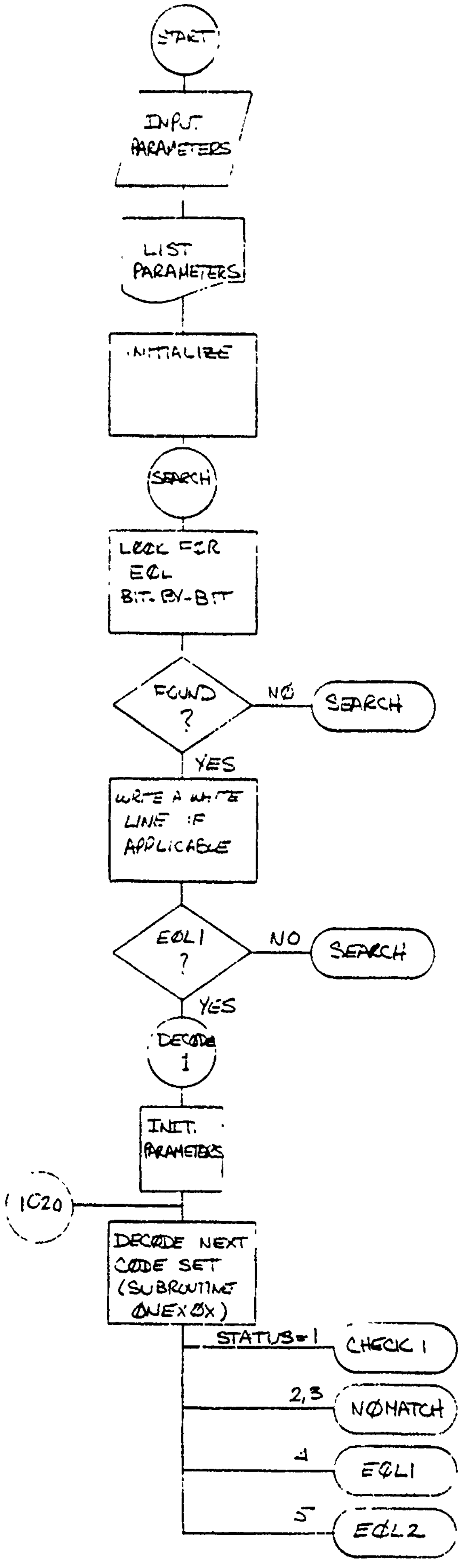
APPENDIX M

PROGRAM FLOW CHART

FOR XEROX ALGORITHM

x 1,5

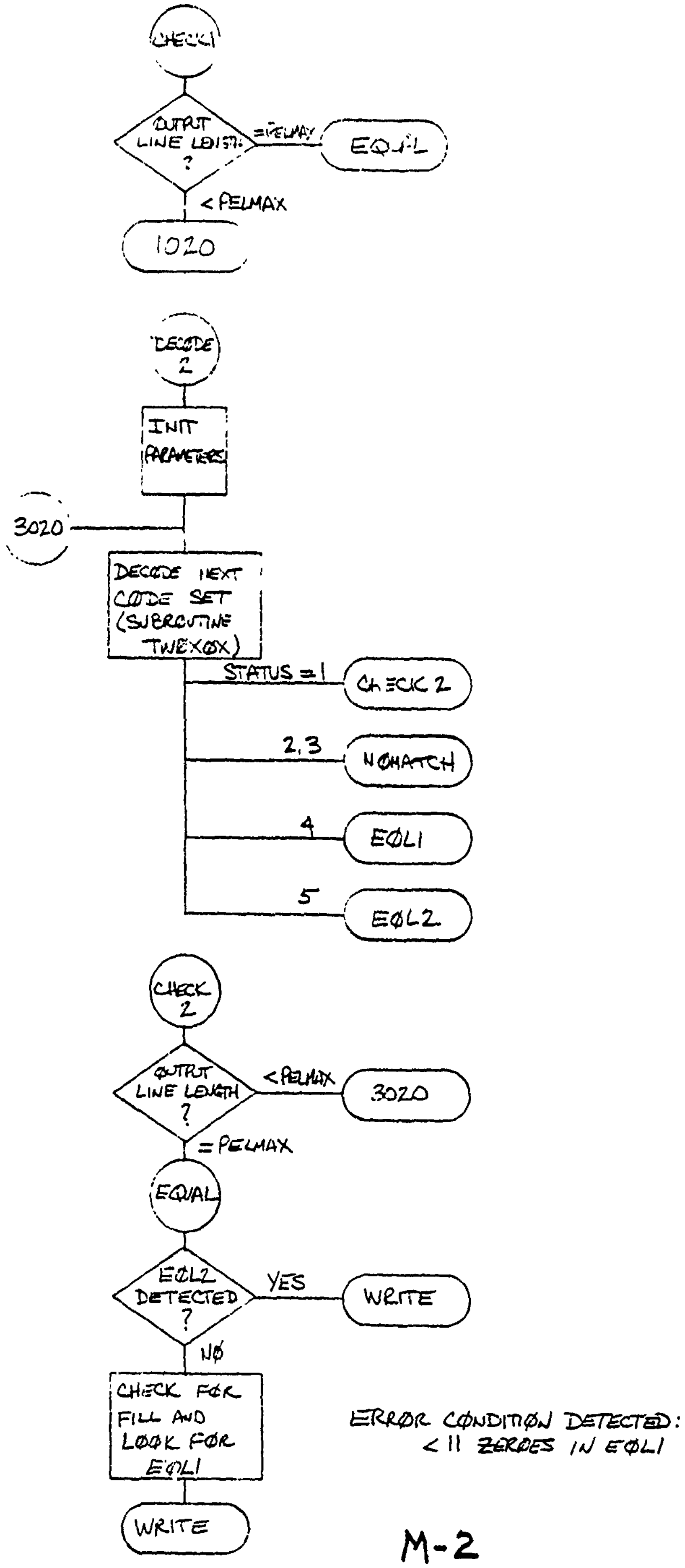
XEROX

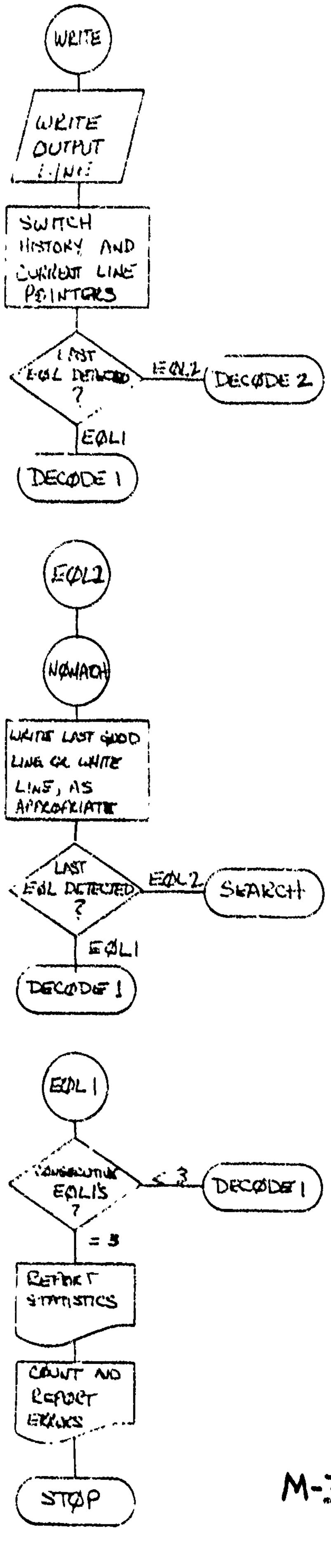


OUTPUT LINE ≤ PELMAX

OUTPUT LINE TOO LONG OR NO MATCH FOUND IN CODE TABLE

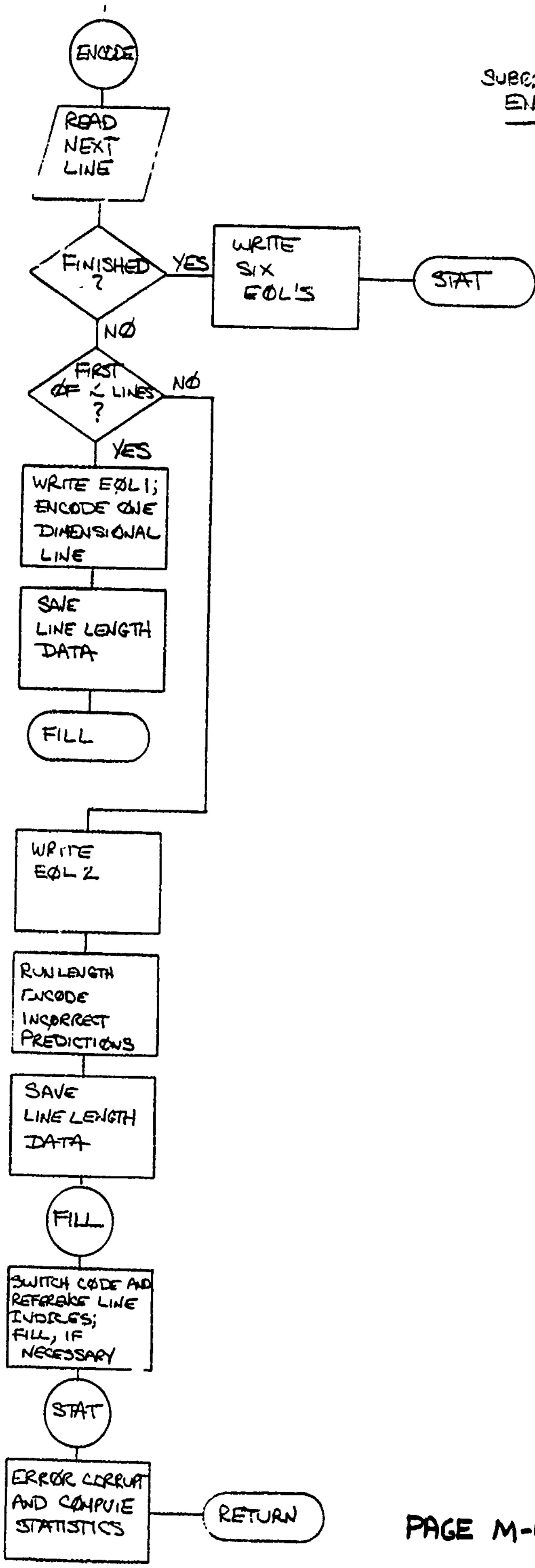
PREMATURE EOL DETECTED


CHECK1
OUTPUT LINE LENGTH ?
=PELMAX
EQUAL
< PELMAX
1020
DECODE 2
INIT PARAMETERS
3020
DECODE NEXT CODE SET (SUBROUTINE TWEXOX)
STATUS = 1
CHECK 2
2,3
NOMATCH
4
EOL1
5
EOL2
CHECK 2
OUTPUT LINE LENGTH ?
< PELMAX
3020
= PELMAX
EQUAL
EOL2 DETECTED ?
YES
WRITE
NO
CHECK FOR FILL AND LOOK FOR EOL1
WRITE
ERROR CONDITION DETECTED:
< 11 ZEROES IN EOL1

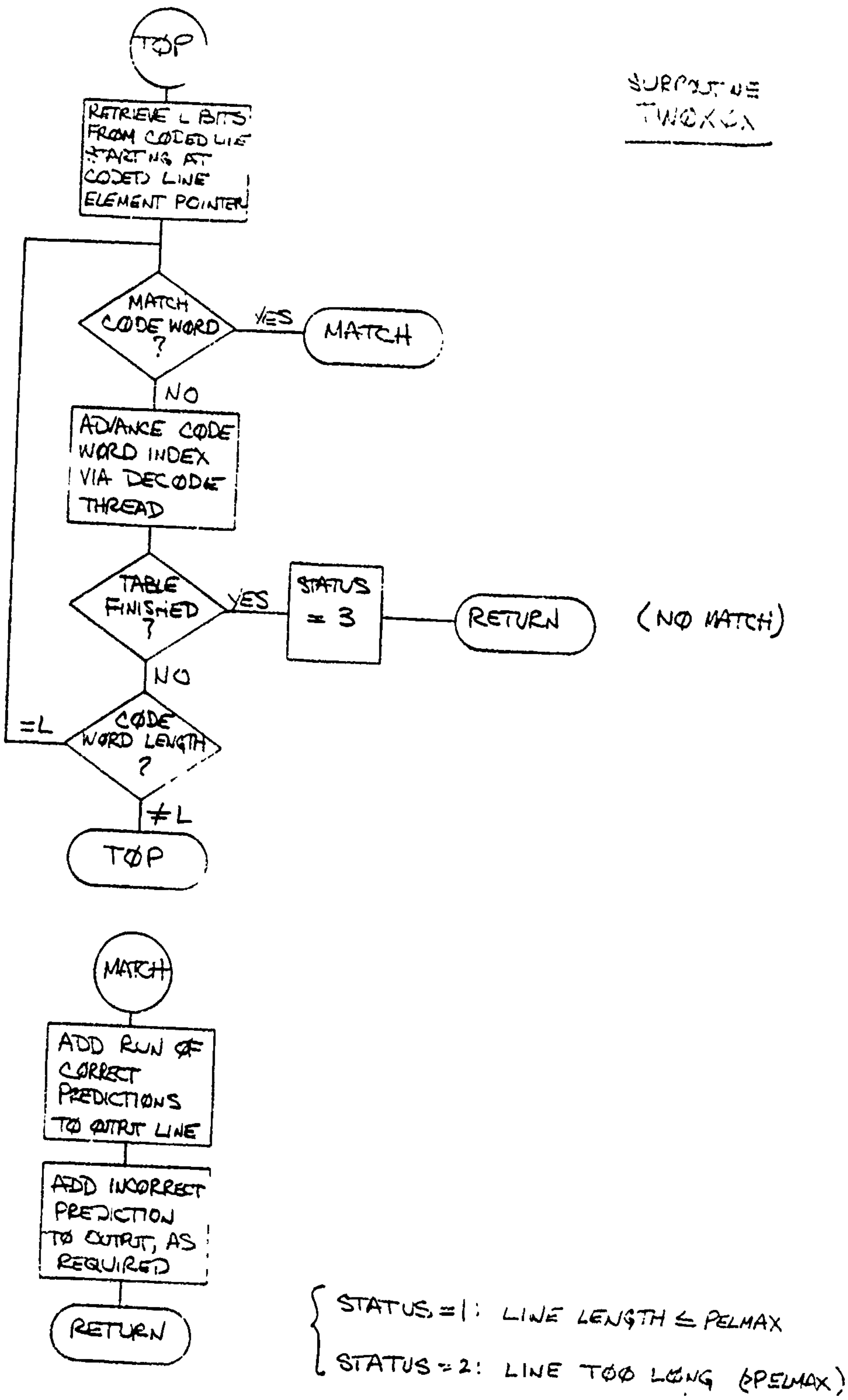

WRITE
WRITE OUTPUT LINE
SWITCH HISTORY AND CURRENT LINE POINTERS
LAST EØL DETECTED ?
EØL2
DECØDE 2
EØL1
DECØDE 1
EØL2
NØMATCH
WRITE LAST GOOD LINE OR WHITE LINE, AS APPROPRIATE
LAST EØL DETECTED ?
EØL2
SEARCH
EØL1
DECØDE 1
EØL 1
CONSECUTIVE EØL'S ?
< 3
DECØDE 1
= 3
REPORT STATISTICS
COUNT AND REPORT ERRORS
STØP

x 4/5

SUBROUTINE ENCØDX



SUBROUTINE
TWØX[illegible]



APPENDIX N

COMPUTER PROGRAM CODE LISTING

XEROX ALGORITHM

```
START OF DCEC UPRINT PROGRAM          DSNAME=00031.XEROX.FORT
C       PROGRAM XEROX
        IMPLICIT INTEGER(A-Z)
        REAL CF3,CF4,ERRATE
C******* LABELED COMMON /G32BIT/ *******
C
        COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
        INTEGER MASK,COMASK,LIBIT,LZBIT
C
        COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *              OTBUF(60,2),STFBUF(240), STAT(3000)
        COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
        COMMON/ERAY/ERRORS(2500)
C********************** FILE DEFINITIONS *******************
C
        COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
        COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
        COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNOBF,BICNT,
     *              INCOD,INREF,OTCOD,OTREF,STFBIT
        COMMON/ICHAR/DD,II,MM,TT,NN,YY
        COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
        LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C     READ INPUT PARAMETERS
   90 WRITE(TERM,100)
  100 FORMAT('$PARAMETERS: INPUT(=I), OR DEFAULT(=D)?')
      READ(TERM,110,ERR=90) INSW
  110 FORMAT(A1)
      IF (INSW.EQ.DD) GO TO 315
      IF (INSW.NE.II) GO TO 90
C
C     READ DIAGNOSTIC SWITCH
C
  114 WRITE(TERM,115)
  115 FORMAT('$DIAGNOSTIC PRINTOUT? (Y OR N): ')
      READ(TERM,110) INSW
      IF(INSW.EQ.YY) GO TO 116
      IF(INSW.EQ.NN) GO TO 120
      GO TO 114
  116 CONTINUE
      DIAG=.TRUE.
C
C     READ MAXIMUM NUMBER OF PELS PER LINE
C
  120 CONTINUE
      WRITE(TERM,130)
  130 FORMAT('$ENTER MAXIMUM NUMBER OF PELS PER LINE: ')
      READ(TERM,140,ERR=120) PELMAX
  140 FORMAT(I4)
      IF(PELMAX.GE.1.AND.PELMAX.LE.1728) GO TO 160
      WRITE(TERM,150) PELMAX
  150 FORMAT('0NUMBER OUT OF RANGE (=',I6,')')
      GO TO 120
C
C     READ VERTICAL SAMPLING
C
  160 CONTINUE
      WRITE(TERM,170)
  170 FORMAT('$ENTER VERTICAL SAMPLING: ')
      READ(TERM,180,ERR=160) VRES
  180 FORMAT(I2)
      IF(VRES.GE.1.AND.VRES.LE.10) GO TO 190
      WRITE(TERM,150) VRES
      GO TO 160
C
C     READ PARAMETER K
C
  190 CONTINUE
      WRITE(TERM,192)
  192 FORMAT('$ENTER PARAMETER K: ')
      READ(TERM,140,ERR=190) K
      IF(K.GE.1.AND.K.LE.3000) GO TO 200
      WRITE(TERM,150) K
      GO TO 190
C
C     READ ERROR PATTERN PHASE
C
```

```
  200 CONTINUE
      WRITE(TERM,210)
  210 FORMAT('$ENTER ERROR PATTERN PHASE: ')
      READ(TERM,220,ERR=200) EPHASE
  220 FORMAT(I1)
      IF(EPHASE.GE.0.AND.EPHASE.LE.3) GO TO 240
      WRITE(TERM,150) EPHASE
      GO TO 200
C
C   READ MINIMUM COMPRESSED LINE LENGTH
C
  240 CONTINUE
      WRITE(TERM,250)
  250 FORMAT('$ENTER MINIMUM COMPRESSED LINE LENGTH: ')
      READ(TERM,140,ERR=240) CMPMAX
      IF(CMPMAX.GE.0.AND.CMPMAX.LE.1728) GO TO 320
      WRITE(TERM,150) CMPMAX
      GO TO 240
C
C   READ NUMBER OF SCAN LINES TO BE PROCESSED
  320 CONTINUE
      WRITE(TERM,330)
  330 FORMAT('$NUMBER OF SCAN LINES TO BE PROCESSED=? ')
      READ(TERM,140,ERR=320) LINMAX
      IF(LINMAX.GE.1.AND.LINMAX.LE.3000) GO TO 280
      WRITE(TERM,150) LINMAX
      GO TO 320
C
C   READ ERROR MODE
C
  280 CONTINUE
      WRITE(TERM,290)
  290 FORMAT('$ERROR MODE=? (M=MANUAL,T=TAPE,N=NO ERRORS)')
      READ(TERM,110,ERR=280) ERRMOD
      IF(ERRMOD.EQ.MM) GO TO 300
      IF(ERRMOD.EQ.TT) GO TO 315
      IF(ERRMOD.NE.NN) GO TO 280
      GO TO 350
C
C   READ ERROR LOCATIONS
C
  300 CONTINUE
      ERRLIM=1
  305 READ(TERM,140) ERRORS(ERRLIM)
      IF(ERRORS(ERRLIM).EQ.9999) GO TO 310
      ERRLIM=ERRLIM+1
      GO TO 305
  310 CONTINUE
      ERRLIM=ERRLIM-1
      GO TO 350
C
C   READ ERROR TAPE FILE AND OPEN
C
  315 CONTINUE
C
      ERRLIM=1
      READ(ERFIL,318,END=317) ERRORS(ERRLIM)
      ERRLIM=ERRLIM+1
  316 READ(ERFIL,318,END=317) ERRORS(ERRLIM)
  318 FORMAT(I16)
      ERRORS(ERRLIM)=ERRORS(ERRLIM)+ERRORS(ERRLIM-1)
      ERRLIM=ERRLIM+1
      GO TO 316
  317 ERRLIM=ERRLIM-1
C
  350 CONTINUE
C
  360 CONTINUE
C   WRITE INPUT PARAMETERS
C
      WRITE(LPFIL,400) PELMAX,VRES,K,EPHASE,CMPMAX,LINMAX
  400 FORMAT('1INPUT PARAMETERS:'/
     *       '0MAXIMUM NUMBER OF PELS PER LINE=',I6/
     *       '0VERTICAL SAMPLING: N=',I4/
     *       '0PARAMETER K =',I4/
     *       '0ERROR PATTERN PHASE =',I4/
     *       '0MINIMUM COMPRESSED LINE LENGTH =',I4,' BITS'/
     *       '0NUMBER OF SCAN LINES TO BE PROCESSED =',I6)
      IF(ERRMOD.EQ.NN) WRITE(LPFIL,410)
  410 FORMAT ('0NO ERRORS INSERTED')
      IF(ERRMOD.EQ.MM) WRITE(TERM,140) (ERRORS(I),I=1,ERRLIM)
      IF(ERRMOD.EQ.TT) WRITE(TERM,420) ERRLIM
```

```
  420 FORMAT(I12,' ERRORS OBTAINED FROM ERROR TAPE')
C******************** BEGIN PROGRAM ************************************
C
C    INITIALIZE
C
      TCDEL=0
      TCDATA=0
      ERRPNT=1
      ERRCNT=0
      INLNCT=0
      ERROFF=EPHASE*1024
      CDELCT=32
      OTELP=1
      CDELP=32+1
      CONSEC=1
      INREF=1
      INCOD=2
      OTREF=1
      OTCOD=2
      STFBIT=0
      BICNT=0
      PREBUF=0
C
      DO 800 I=1,240
      STFBUF(I)=0
      CDBUF(I)=0
  800 CONTINUE
      DO 850 I=1,60
      OTBUF(I,OTREF)=0
      OTBUF(I,OTCOD)=0
      PELBUF(I,INREF)=0
      PELBUF(I,INCOD)=0
  850 CONTINUE
      SEARCH=.TRUE.
      SYNC=.FALSE.
      WRITE=.FALSE.
C
C    SEARCH MODE: LOOK FOR EOL1 BIT-BY-BIT
C
 900  CONTINUE
      CALL GETLX(12,MODE,LBITS,L)
      GO TO (910,930,910,920),MODE
      STOP 900
 910  CONTINUE
C
C    EOL1 NOT FOUND; ADVANCE POINTER AND TRY AGAIN
C
      CDELP=CDELP+1
      GO TO 900
 920  CONTINUE
      STOP 920
 930  CONTINUE
C
C    EOL1 FOUND
C
      SEARCH=.FALSE.
      CDELP=CDELP+L
      IF(WRITE) GO TO 935
      WRITE=.TRUE.
      GO TO 960
 935  CONTINUE
C
C    SET OUTPUT DECODE LINE TO 0 AND WRITE OUT
      DO 950 I=1,60
      OTBUF(I,OTCOD)=0
 950  CONTINUE
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
 960  CONTINUE
      IF(MODE-2)965,1000,900
 965  STOP 965
 1000 CONTINUE
C
C    PERFORM ONE-DIMENSIONAL DECODE OF A COMPLETE LINE
C    FIRST,SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 1010 I=1,60
      OTBUF(I,OTCOD)=0
 1010 CONTINUE
C
      INDEX=3
```

```
      COLOR=1
      OTELP=1
C
 1020 CONTINUE
      CALL ONEXOX(INDEX,COLOR,STATUS,L)
      GO TO (1030,1070,1070,1035,1040),STATUS
C              1    2    3    4    5
      STOP 1000
C
C   RUN ADDED; CHECK LENGTH OF OUTPUT LINE
C
 1030 CONTINUE
      ONE=.TRUE.
      IF(OTELP-1-PELMAX) 1031,3032,1050
 1031 CONTINUE
      IF(CHCOL)COLOR=MOD(COLOR+2,2)+1
      INDEX=3
      GO TO 1020
 3000 CONTINUE
C
C   PERFORM TWO-DIMENSIONAL DECODE
C
C
C   FIRST,SET OUTPUT BUFFER TO WHITE
C   (ONLY BLACK RUNS WILL BE INSERTED)
C
      DO 3010 I=1,60
      OTBUF(I,OTCOD)=0
 3010 CONTINUE
C
      INDEX=1
      COLOR=1
      OTELP=1
C
 3020 CONTINUE
      CALL TWOXOX(INDEX,COLOR,STATUS,L)
      GO TO (3030,1070,1070,1035,1040),STATUS
C              1    2    3    4    5
      STOP 3000
C
C   RUN ADDED; LOOK FOR NEXT RUN
C
 3030 CONTINUE
      ONE=.FALSE.
      IF(OTELP-1-PELMAX) 3031,3032,1050
 3031 CONTINUE
      INDEX=1
      GO TO 3020
C
C   LINELENGTH = PELMAX; FIRST CHECK FOR EOL2
C
 3032 CONTINUE
      CALL GETLX(5,MODE,LBITS,L)
      GO TO (3034,1050,1050,1050),MODE
 3034 CONTINUE
C
      IF(LBITS.EQ.CODERD(3,92)) GO TO 1059
C
C      NOT AN EOL2
C
C   LINE LENGTH=PELMAX; CHECK FOR FILL AND LOOK FOR EOL1
C
 1032 CONTINUE
      ZERO=-1
 1033 CONTINUE
      ZERO=ZERO+1
      CALL GETLX(1,MODE,LBITS,L)
C
      GO TO (1034,1050,1050,1050),MODE
C
C - CHECK FOR FILL
C
 1034 CONTINUE
C
      CDELP=CDELP+L
      IF(LBITS.EQ.0) GO TO 1033
      IF(ZERO.LE.10) GO TO 1070
C
C   EOL1 FOUND
C
      MODE=2
      GO TO 1060
```

```
C
C   PREMATURE EOL DETECTED
C
C
C   EOL1 DETECTED
C
 1035 CONTINUE
      CDELP=CDELP+L
      STATUS=4
      IF(OTELP.LE.5) CONSEC=CONSEC+1
      IF(CONSEC-2)1080,1000,2000
C
C  EOL 2 DETECTED
C
 1040 CONTINUE
      CDELP=CDELP+L
      STATUS=5
C
      GO TO 1080
C
C   PROBLEMS,PROBLEMS
C
 1050 STOP 1050
C
C   LINE LENGTH CORRECT. EOL DETECTED PROPERLY; WRITE OUTPUT LINE
C
 1059 MODE=3
      CDELP=CDELP+L
 1060 CONTINUE
      WRITE(OTFIL)OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
      CONSEC=1
      IF(ONE) SYNC=.TRUE.
      TEMP=OTREF
      OTREF=OTCOD
      OTCOD=TEMP
      IF(MODE.EQ.2) GO TO 1000
      GO TO 3000
C
C   LINE TOO LONG OR NO MATCH
C
 1070 CONTINUE
      WRITE=.FALSE.
C
C   LINE SHORT
C
 1080 CONTINUE
      IF(.NOT.SYNC) GO TO 1090
C
C   WRITE LAST GOOD LINE
C
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTREF),I=1,60)
      SYNC=.FALSE.
      GO TO 1110
 1090 CONTINUE
C
C   WRITE A WHITE LINE
C
      DO 1100 I=1, 60
 1100 OTBUF(I,OTCOD)=0
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
 1110 OTLNNO=LNNOBF
      IF(STATUS.EQ.4) GO TO 1000
      SEARCH=.TRUE.
      GO TO 900
C
C   END OF MESSAGE
C
 2000 CONTINUE
      WRITE(LPFIL,2010) CONSEC
 2010 FORMAT('0END OF MESSAGE DETECTED (',I2,' EOL''S)')
C
C   REPORT COMPRESSION FACTOR, ERROR SENSITIVITY FACTOR,BIT ERROR RATE
C
      ERRATE=FLOAT(ERRCNT)/FLOAT(TCDEL)
      WRITE(LPFIL,2020) TCDEL,TCDATA,INLNCT,ERRATE
 2020 FORMAT('0TOTAL NUMBER OF CODED BITS = ',I8/
     *       '0TOTAL NUMBER OF CODED DATA BITS = ',I8/
     *       '0TOTAL NUMBER OF INPUT LINES PROCESSED = ',I8/
     *       '0BIT ERROR RATE = ',G14.6)
C
      CALL STATS(STAT,INLNCT,DIAG)
```

```
      CF3=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDEL)
      CF4=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDATA)
C
      WRITE(LPFIL,2030) CF3,CF4
 2030 FORMAT('0COMPRESSION FACTOR FOR G3 MACHINE (CF3) =',F8.4/
     *       '0COMPRESSION FACTOR FOR G4 MACHINE (CF4) =',F8.4)
C
      CALL ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      STOP
      E N D
      SUBROUTINE GETLX(LBITS,MODE,WRD,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *            OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNCBF,BICNT,
     *            INCOD,INREF,OTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C*************************** BEGIN PROGRAM ***************************
C
      MODE=4
C
C   RETRIEVE NEXT BIT FROM CDBUF
C
 100  CONTINUE
C
C   ENCODE A NEW LINE IF NECESSARY
C
      IF(LBITS+CDELP-1.LE.CDELCT) GO TO 200
      IF(CDELCT-CDELP+1) 170,190,180
  170 STOP 170
  180 CONTINUE
      STFBUF(1)=I48(STFBUF,CDELP,CDELCT-CDELP+1)
  190 CONTINUE
      CDELP=32-(CDELCT-CDELP)
      CALL ENCODX
 200  CONTINUE
      WRD=I48(STFBUF,CDELP,LBITS)
      L=LBITS
      IF(L.EQ.12.AND.WRD.EQ.CODERD(3,93))GO TO 300
  250 CONTINUE
      MODE=1
      RETURN
  300 CONTINUE
      MODE=2
      RETURN
      END
      SUBROUTINE ENCODX
C
      IMPLICIT INTEGER(A-Z)
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *            OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
```

```
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNOBF,B1CNT,
     *              INCOD,INREF,CTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C************************************ BEGIN PROGRAM ********************
C
C    INITIALIZE VARIABLES
C
      CDELCT=32
      CDDATA=0
      DO 50 I=2,240
      CDBUF(I)=0
      STFBUF(I)=0
   50 CONTINUE
C
C    READ INPUT PICTURE FILE
C
  100 CONTINUE
      READ(PELFIL,END=120,ERR=500)
     * INLNNO,INELCT,(PELBUF(I,INCOD),I=1,60)
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
      IF(INELCT.LT.PELMAX) CALL EXIT
      INLNCT=INLNCT+1
C
C    LOAD OUTPUT LINE NUMBER BUFFER
C
      LNNOBF=INLNNO
      IF(SEARCH)OTLNNO=LNNOBF
C
      IF(INLNNO.LE.LINMAX) GO TO 140
C
C    WRITE SIX EOL1'S
C
  120 CONTINUE
      DO 130 I=1,6
      CALL CODEX(0,1,CDELCT,CDDATA)
  130 CONTINUE
      DO 135 I=1,6
      STFBUF(I)=CDBUF(I)
  135 CONTINUE
      GO TO 400
C
C    FIRST OF K LINES?
C
  140 CONTINUE
      IF(MOD(INLNCT-1,K).NE.0) GO TO 600
C
C    ONE-DIMENSIONAL CODING
C    WRITE ONE EOL1
C
      B1CNT=0
      CALL CODEX(0,1,CDELCT,CDDATA)
C
      POLAR=1
C
C    TEST COLOR OF FIRST ELEMENT
C
      IF(I48(PELBUF(1,INCOD),1,1).EQ.0) GO TO 150
C
C    FIRST ELEMENT BLACK; ENCODE 0-LENGTH WHITE RUN
C
      CALL XCODLR(0,1,CDELCT,CDDATA)
      POLAR=2
C
C    CALCULATE RUN LENGTH AND ENCODE
C
  150 CONTINUE
      RUN=0
      DO 200 I=1,PELMAX
      PEL=I48(PELBUF(1,INCOD),I,1)+1
      IF(PEL.EQ.POLAR) GO TO 180
      CALL XCODLR(RUN,POLAR,CDELCT,CDDATA)
      IF(.NOT.DIAG) GO TO 170
      WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
  160 FORMAT(4I8)
  170 CONTINUE
      RUN=1
      POLAR=MOD(POLAR+2,2)+1
      GO TO 200
```

```
180 CONTINUE
    RUN=RUN+1
200 CONTINUE
    CALL XCODLR(RUN,POLAR,CDELCT,CDDATA)
    BICNT=CDELCT
C
C   SAVE LINE LENGTH(DATA BITS + EOL)
C
    STAT(INLNCT)=CDDATA+CODERD(1,93)
    IF(.NOT.DIAG) GO TO 210
    WRITE(TERM,160) RUN,POLAR,CDELCT,CDDATA
    GO TO 210
C
C   TWO-DIMENSIONAL CODING
C
600 CONTINUE
C
C   WRITE ONE EOL2
C
    CALL CODEX(0,2,CDELCT,CDDATA)
C
    RUN=0
    DO 700 I=1,PELMAX
    PEL2=I48(PELBUF(1,INCOD),I-2,2)
    PEL1=I48(PELBUF(1,INREF),I-2,5)
    CALL MI23(PEL1,PEL2,32-6,5)
    PEL2P1=PEL2+1
    IF(PREDCT(PEL2P1).EQ.I48(PELBUF(1,INCOD),I,1))GO TO 680
    CALL CODEX(RUN,0,CDELCT,CDDATA)
    IF(.NOT.DIAG)GO TO 670
    WRITE(TERM,160)RUN,PEL2,CDELCT,CDDATA
670 CONTINUE
    RUN=0
    GO TO 700
680 CONTINUE
    RUN=RUN+1
700 CONTINUE
C
C   SPECIAL CASE IF LAST PEL ON LINE IS PREDICTED CORRECTLY;
C   FORCE AN INCORRECT PREDICTION AT PELMAX + 1
C
    IF(RUN) 705,740,710
705 STOP 705
710 CALL CODEX(RUN,0,CDELCT,CDDATA)
    IF(DIAG) WRITE(TERM,160) RUN,PEL2,CDELCT,CDDATA
740 CONTINUE
    BICNT=BICNT+CDELCT
C
C   SAVE LINE LENGTH(DATA BITS PLUS EOL)
C
    STAT(INLNCT)=CDDATA+CODERD(1,92)
210 CONTINUE
C
C     SWITCH CODE & REFERENCE LINES
C
    TEMP=INREF
    INREF=INCOD
    INCOD=TEMP
C
C   TRANSFER CDBUF TO STFBUF
C
    CDELW=(CDELCT+32-1)/32
    DO 240 I=2,CDELW
    STFBUF(I)=CDBUF(I)
240 CONTINUE
C
C   CHECK CODED LINE LENGTH
C
    FILL=0
    IF(MOD(INLNCT,K).EQ.0)FILL=K*(CMPMAX+32)-BICNT
    IF(FILL) 400,400,250
C
C   CODE LINE TOO SHORT; FILL IT TO CMPMAX
250 CONTINUE
    CDELCT=CDELCT+FILL
C
C   ACCUMULATE STATISTICS AND ERROR CORRUPT
C
400 CONTINUE
    IF(ERRMOD.EQ.NN) GO TO 390
C
C   ERROR CORRUPT
```

```
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C    ERROR IN RANGE OF CODED LINE; CHANGE APPROPFIATE BIT
C
      BIT=I48(STFBUF,ERRBIT+32,1)
      BIT=MOD(BIT+1,2)
      CALL MI28(BIT,STFBUF,ERRBIT+32,1)
      ERRCNT=ERRCNT+1
C
C    INCREMENT ERROR LIST POINTER
C
  360 CONTINUE
      ERRPNT=ERRPNT+1
      IF(ERRPNT.LE.ERRLIM) GO TO 350
C
C    ERROR LIST EXHAUSTED
C
      ERRPNT=ERRPNT-1
      WRITE(LPFIL,370) ERRPNT,ERRORS(ERRPNT)
  370 FORMAT('0ERROR LIST EXHAUSTED AT',I10,'TH ERROR;'/
     *       ' LAST ERROR OCCURRED AT',I10,' BITS')
      ERRMOD=NN
C
C    COMPUTE STATISTICS
C
  390 CONTINUE
      TCDEL=TCDEL+CDELCT-32
      TCDATA=TCDATA+CDDATA
      IF(DIAG) WRITE(TERM,160) INLNCT, CDDATA
C
      IF (.NOT.DIAG) GO TO 460
      CDELW=(CDELCT+32-1)/32
      WRITE(LPFIL,450) (CDBUF(I),I=1,CDELW)
      WRITE(LPFIL,450) (STFBUF(I),I=1,CDELW)
  450 FORMAT(6Z12)
  460 CONTINUE
      RETURN
C
  500 CONTINUE
      CALL EXIT
C
      END
      SUBROUTINE CODEX(LENGTH,MODE,CDELCT,CDDATA)
C
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *               OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C
C********************************** BEGIN PROGRAM **********************
C
C    INITIALIZE MAKE UP CODE, MAKE UP CODE LENGTH
C
      MCODE=0
      MLENG=0
C
C    CHECK INPUTS
C
      IF(MODE.LT.0.OR.MODE.GT.2) CALL EXIT
      IF(LENGTH.LT.0.OR.LENGTH.GT.1728) CALL EXIT
      IF(MODE.GT.0) GO TO 50
C
      IF(LENGTH.LE.63) GO TO 10
C
C    CALCULATE MAKE UP CODE INDEX, CODE, LENGTH
C     AND WRITE TO CODE LINE
C
      INDEX=LENGTH/64+64
      MCODE=CODERD(3,INDEX)
      MLENG=CODERD(1,INDEX)
      CALL MI28(MCODE,CDBUF,CDELCT+1,MLENG)
      CDELCT=CDELCT+MLENG
      CDDATA=CDDATA+MLENG
C
C    CALCULATE TERMINATING CODE INDEX, CODE, LENGTH
C     AND ADD TO CODE LINE
C
```

```
   10 CONTINUE
      INDEX=MOD(LENGTH,64)+1
      TCODE=CODERD(3,INDEX)
      TLENG=CODERD(1,INDEX)
      CALL MI2B(TCODE,CDBUF,CDELCT+1,TLENG)
      CDELCT=CDELCT+TLENG
      CDDATA=CDDATA+TLENG
C
      RETURN
C
C    ADD EOL TO CODE LINE
C
   50 CONTINUE
      MODEP=94-MODE
      CALL MI2B(CODERD(3,MODEP),CDBUF,CDELCT+1,CODERD(1,MODEP))
      CDELCT=CDELCT+CODERD(1,MODEP)
C
      RETURN
C
      E N D
      SUBROUTINE ONEXOX(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *              OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C************************ FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNOBF,BICNT,
     *              INCOD,INREF,OTCOD,OTREF,STFBIT
      COMMON/ICHAR/OD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C***************************** BEGIN PROGRAM ***************************
C
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODE(1,INDEX,COLOR)
      CALL GETLX(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z8,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODE(3,INDEX,COLOR)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODE(2,INDEX,COLOR)
      IF(INDEX.GE.93) GO TO 1190
      IF(CODE(1,INDEX,COLOR).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
C
C
C    TEST FOR MAKE UP OR TERMINATING CODE
C
      RUNLEN=INDEX-1
```

```
      IF(INDEX.GE.65) RUNLEN=(INDEX-64)*64
      IF(RUNLEN.EQ.0) GO TO 1160
      IF(COLOR.EQ.1) GO TO 1155
      IF(RUNLEN.LT.0) STOP 1100
C
C    ADD BLACK RUN TO OUTPUT BUFFER
C
      DO 1150 I=1,RUNLEN
      CALL MI2U(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C    ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C OUTPUT LINE LESS THAN OR EQUAL TO MAX SPECIFIED
C
 1160 CONTINUE
      IF(INDEX.LT.65) GO TO 1170
      INDEX=3
      GO TO 1000
C
C    RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1170 CONTINUE
      CHCOL=.TRUE.
      STATUS=1
      RETURN
C
C    RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C    NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C    EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C    EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      SUBROUTINE TWOXOX(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *            OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C************************* FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C*********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,BICNT,
     *            INCOD,INREF,OTCOD,CTREF,STFBIT
```

```
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODERD(1,INDEX)
      CALL GETLX(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,2I2,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODERD(3,INDEX)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODERD(2,INDEX)
      IF(INDEX.GE.94) GO TO 1190
      IF(CODERD(1,INDEX).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
C
C    TEST FOR MAKE UP OR TERMINATING CODE
C
      RUNLEN=INDEX-1
      IF(INDEX.GE.65)RUNLEN=(INDEX-64)*64
      IF(RUNLEN)1110,1155,1140
 1110 STOP 1110
C
C    ADD RUN OF CORRECT PREDICTIONS TO OUTPUT LINE
C
 1140 CONTINUE
      DO 1150 I=1,RUNLEN
      PEL2=I4B(OTBUF(1,OTCOD),OTELP-2,2)
      PEL1=I4B(OTBUF(1,OTREF),OTELP-2,5)
      CALL MI2B(PEL1,PEL2,32-6,5)
      PEL2P1=PEL2+1
      CALL MI2B(PREDCT(PEL2P1),OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      IF(INDEX.LT.65)GO TO 1155
      INDEX=1
      GO TO 1000
C
C    ADD INCORRECT PREDICTION TO OUTPUT LINE
C
 1155 CONTINUE
      IF(OTELP.EQ.PELMAX+1) GO TO 1160
      PEL2=I4B(OTBUF(1,OTCOD),OTELP-2,2)
      PEL1=I4B(OTBUF(1,OTREF),OTELP-2,5)
      CALL MI2B(PEL1,PEL2,32-6,5)
      PEL2P1=PEL2+1
      PEL=PREDCT(PEL2P1)
      PEL=MOD(PEL+1,2)
      CALL MI2B(PEL,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OT[illegible]+1
      IF(OTELP-[illegible]T.PELMAX)GO [illegible] 1180
C
C    RUN ADDED [illegible] OUTPUT LINE[illegible]TH LESS THAN OR EQUAL TO PELMAX (1)
C
 11[illegible]0 CONTINUE
      STATUS=1
      RETURN
C
C    [illegible] ADDE[illegible] PELMAX EXC[illegible]ED; LINE [illegible] LONG (2)
C
 11[illegible]0 CONTINUE
      [illegible]DIAG[illegible] (TERM,11[illegible] (OTBUF([illegible]CD),I=1,60)
 1[illegible]
```

```
      STATUS=2
      RETURN
C
C   NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      STATUS=3
      RETURN
C
C   EOL1 DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C   EOL2 DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D

      BLOCK DATA
C
      IMPLICIT INTEGER(A-Z)
C************************ FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),
     *              OTBUF(60,2),STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C********************* LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *              CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *              ERRCNT,INLNCT,CONSEC,LNNCBF,BICNT,
     *              INCOD,INREF,OTCOD,OTREF,STFBIT
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE
C
      DATA TERM,LPFIL,PELFIL,OTFIL,ERFIL/5,6,1,2,3/
      DATA DD,II,MM,TT,NN,YY/'D','I','M','T','N','Y'/
      DATA PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX/1728,2,0,96,'T',3000
      DATA K/2/
      DATA DIAG/.FALSE./
       DATA CODE(1,  1,1),CODE(2,  1,1),CODE(3,  1,1)/ 8, 70,Z0035/
       DATA CODE(1,  2,1),CODE(2,  2,1),CODE(3,  2,1)/ 6, 90,Z0007/
       DATA CODE(1,  3,1),CODE(2,  3,1),CODE(3,  3,1)/ 4,  4,Z0007/
       DATA CODE(1,  4,1),CODE(2,  4,1),CODE(3,  4,1)/ 4,  5,Z0006/
       DATA CODE(1,  5,1),CODE(2,  5,1),CODE(3,  5,1)/ 4,  6,Z0008/
       DATA CODE(1,  6,1),CODE(2,  6,1),CODE(3,  6,1)/ 4,  7,Z000C/
       DATA CODE(1,  7,1),CODE(2,  7,1),CODE(3,  7,1)/ 4,  8,Z000E/
       DATA CODE(1,  8,1),CODE(2,  8,1),CODE(3,  8,1)/ 4,  9,Z000F/
       DATA CODE(1,  9,1),CODE(2,  9,1),CODE(3,  9,1)/ 5, 10,Z0013/
       DATA CODE(1, 10,1),CODE(2, 10,1),CODE(3, 10,1)/ 5, 11,Z0014/
       DATA CODE(1, 11,1),CODE(2, 11,1),CODE(3, 11,1)/ 5, 12,Z0007/
       DATA CODE(1, 12,1),CODE(2, 12,1),CODE(3, 12,1)/ 5, 65,Z0008/
       DATA CODE(1, 13,1),CODE(2, 13,1),CODE(3, 13,1)/ 6, 14,Z0008/
       DATA CODE(1, 14,1),CODE(2, 14,1),CODE(3, 14,1)/ 6, 15,Z0003/
       DATA CODE(1, 15,1),CODE(2, 15,1),CODE(3, 15,1)/ 6, 16,Z0034/
       DATA CODE(1, 16,1),CODE(2, 16,1),CODE(3, 16,1)/ 6, 17,Z0035/
       DATA CODE(1, 17,1),CODE(2, 17,1),CODE(3, 17,1)/ 6, 18,Z002A/
       DATA CODE(1, 18,1),CODE(2, 18,1),CODE(3, 18,1)/ 6, 19,Z002B/
       DATA CODE(1, 19,1),CODE(2, 19,1),CODE(3, 19,1)/ 7, 20,Z0027/
       DATA CODE(1, 20,1),CODE(2, 20,1),CODE(3, 20,1)/ 7, 21,Z000C/
       DATA CODE(1, 21,1),CODE(2, 21,1),CODE(3, 21,1)/ 7, 22,Z0008/
       DATA CODE(1, 22,1),CODE(2, 22,1),CODE(3, 22,1)/ 7, 23,Z0017/
       DATA CODE(1, 23,1),CODE(2, 23,1),CODE(3, 23,1)/ 7, 24,Z0003/
       DATA CODE(1, 24,1),CODE(2, 24,1),CODE(3, 24,1)/ 7, 25,Z0004/
       DATA CODE(1, 25,1),CODE(2, 25,1),CODE(3, 25,1)/ 7, 26,Z0028/
       DATA CODE(1, 26,1),CODE(2, 26,1),CODE(3, 26,1)/ 7, 27,Z002B/
       DATA CODE(1, 27,1),CODE(2, 27,1),CODE(3, 27,1)/ 7, 28,Z0013/
       DATA CODE(1, 28,1),CODE(2, 28,1),CODE(3, 28,1)/ 7, 29,Z0024/
       DATA CODE(1, 29,1),CODE(2, 29,1),CODE(3, 29,1)/ 7, 68,Z0018/
       DATA CODE(1, 30,1),CODE(2, 30,1),CODE(3, 30,1)/ 8, 31,Z0002/
       DATA CODE(1, 31,1),CODE(2, 31,1),CODE(3, 31,1)/ 8, 32,Z0003/
       DATA CODE(1, 32,1),CODE(2, 32,1),CODE(3, 32,1)/ 8, 33,Z001A/
       DATA CODE(1, 33,1),CODE(2, 33,1),CODE(3, 33,1)/ 8, 34,Z001B/
```

```
DATA CODE(1, 34,1),CODE(2, 34,1),CODE(3, 34,1)/ 8, 35,Z0012/
DATA CODE(1, 35,1),CODE(2, 35,1),CODE(3, 35,1)/ 8, 36,Z0013/
DATA CODE(1, 36,1),CODE(2, 36,1),CODE(3, 36,1)/ 8, 37,Z0014/
DATA CODE(1, 37,1),CODE(2, 37,1),CODE(3, 37,1)/ 8, 38,Z0015/
DATA CODE(1, 38,1),CODE(2, 38,1),CODE(3, 38,1)/ 8, 39,Z0016/
DATA CODE(1, 39,1),CODE(2, 39,1),CODE(3, 39,1)/ 8, 40,Z0017/
DATA CODE(1, 40,1),CODE(2, 40,1),CODE(3, 40,1)/ 8, 41,Z0028/
DATA CODE(1, 41,1),CODE(2, 41,1),CODE(3, 41,1)/ 8, 42,Z0029/
DATA CODE(1, 42,1),CODE(2, 42,1),CODE(3, 42,1)/ 8, 43,Z002A/
DATA CODE(1, 43,1),CODE(2, 43,1),CODE(3, 43,1)/ 8, 44,Z002B/
DATA CODE(1, 44,1),CODE(2, 44,1),CODE(3, 44,1)/ 8, 45,Z002C/
DATA CODE(1, 45,1),CODE(2, 45,1),CODE(3, 45,1)/ 8, 46,Z002D/
DATA CODE(1, 46,1),CODE(2, 46,1),CODE(3, 46,1)/ 8, 47,Z0004/
DATA CODE(1, 47,1),CODE(2, 47,1),CODE(3, 47,1)/ 8, 48,Z0005/
DATA CODE(1, 48,1),CODE(2, 48,1),CODE(3, 48,1)/ 8, 49,Z000A/
DATA CODE(1, 49,1),CODE(2, 49,1),CODE(3, 49,1)/ 8, 50,Z000B/
DATA CODE(1, 50,1),CODE(2, 50,1),CODE(3, 50,1)/ 8, 51,Z0052/
DATA CODE(1, 51,1),CODE(2, 51,1),CODE(3, 51,1)/ 8, 52,Z0053/
DATA CODE(1, 52,1),CODE(2, 52,1),CODE(3, 52,1)/ 8, 53,Z0054/
DATA CODE(1, 53,1),CODE(2, 53,1),CODE(3, 53,1)/ 8, 54,Z0055/
DATA CODE(1, 54,1),CODE(2, 54,1),CODE(3, 54,1)/ 8, 55,Z0024/
DATA CODE(1, 55,1),CODE(2, 55,1),CODE(3, 55,1)/ 8, 56,Z0025/
DATA CODE(1, 56,1),CODE(2, 56,1),CODE(3, 56,1)/ 8, 57,Z0058/
DATA CODE(1, 57,1),CODE(2, 57,1),CODE(3, 57,1)/ 8, 58,Z0059/
DATA CODE(1, 58,1),CODE(2, 58,1),CODE(3, 58,1)/ 8, 59,Z005A/
DATA CODE(1, 59,1),CODE(2, 59,1),CODE(3, 59,1)/ 8, 60,Z005B/
DATA CODE(1, 60,1),CODE(2, 60,1),CODE(3, 60,1)/ 8, 61,Z004A/
DATA CODE(1, 61,1),CODE(2, 61,1),CODE(3, 61,1)/ 8, 62,Z004B/
DATA CODE(1, 62,1),CODE(2, 62,1),CODE(3, 62,1)/ 8, 63,Z0032/
DATA CODE(1, 63,1),CODE(2, 63,1),CODE(3, 63,1)/ 8, 64,Z0033/
DATA CODE(1, 64,1),CODE(2, 64,1),CODE(3, 64,1)/ 8, 69,Z0034/
DATA CODE(1, 65,1),CODE(2, 65,1),CODE(3, 65,1)/ 5, 66,Z001B/
DATA CODE(1, 66,1),CODE(2, 66,1),CODE(3, 66,1)/ 5, 67,Z0012/
DATA CODE(1, 67,1),CODE(2, 67,1),CODE(3, 67,1)/ 6,  2,Z0017/
DATA CODE(1, 68,1),CODE(2, 68,1),CODE(3, 68,1)/ 7, 30,Z0037/
DATA CODE(1, 69,1),CODE(2, 69,1),CODE(3, 69,1)/ 8,  1,Z0036/
DATA CODE(1, 70,1),CODE(2, 70,1),CODE(3, 70,1)/ 8, 71,Z0037/
DATA CODE(1, 71,1),CODE(2, 71,1),CODE(3, 71,1)/ 8, 72,Z0064/
DATA CODE(1, 72,1),CODE(2, 72,1),CODE(3, 72,1)/ 8, 73,Z0065/
DATA CODE(1, 73,1),CODE(2, 73,1),CODE(3, 73,1)/ 8, 74,Z0068/
DATA CODE(1, 74,1),CODE(2, 74,1),CODE(3, 74,1)/ 8, 75,Z0067/
DATA CODE(1, 75,1),CODE(2, 75,1),CODE(3, 75,1)/ 9, 76,Z00CC/
DATA CODE(1, 76,1),CODE(2, 76,1),CODE(3, 76,1)/ 9, 77,Z00CD/
DATA CODE(1, 77,1),CODE(2, 77,1),CODE(3, 77,1)/ 9, 78,Z00D2/
DATA CODE(1, 78,1),CODE(2, 78,1),CODE(3, 78,1)/ 9, 79,Z00D3/
DATA CODE(1, 79,1),CODE(2, 79,1),CODE(3, 79,1)/ 9, 80,Z00D4/
DATA CODE(1, 80,1),CODE(2, 80,1),CODE(3, 80,1)/ 9, 81,Z00D5/
DATA CODE(1, 81,1),CODE(2, 81,1),CODE(3, 81,1)/ 9, 82,Z00D6/
DATA CODE(1, 82,1),CODE(2, 82,1),CODE(3, 82,1)/ 9, 83,Z00D7/
DATA CODE(1, 83,1),CODE(2, 83,1),CODE(3, 83,1)/ 9, 84,Z00D8/
DATA CODE(1, 84,1),CODE(2, 84,1),CODE(3, 84,1)/ 9, 85,Z00D9/
DATA CODE(1, 85,1),CODE(2, 85,1),CODE(3, 85,1)/ 9, 86,Z00DA/
DATA CODE(1, 86,1),CODE(2, 86,1),CODE(3, 86,1)/ 9, 87,Z00DB/
DATA CODE(1, 87,1),CODE(2, 87,1),CODE(3, 87,1)/ 9, 88,Z0098/
DATA CODE(1, 88,1),CODE(2, 88,1),CODE(3, 88,1)/ 9, 89,Z0099/
DATA CODE(1, 89,1),CODE(2, 89,1),CODE(3, 89,1)/ 9, 91,Z009A/
DATA CODE(1, 90,1),CODE(2, 90,1),CODE(3, 90,1)/ 6, 13,Z0018/
DATA CODE(1, 91,1),CODE(2, 91,1),CODE(3, 91,1)/ 9, 92,Z009B/
DATA CODE(1, 92,1),CODE(2, 92,1),CODE(3, 92,1)/12, 93,Z0001/
DATA CODE(1,  1,2),CODE(2,  1,2),CODE(3,  1,2)/10, 65,Z0037/
DATA CODE(1,  2,2),CODE(2,  2,2),CODE(3,  2,2)/ 3,  6,Z0002/
DATA CODE(1,  3,2),CODE(2,  3,2),CODE(3,  3,2)/ 2,  4,Z0003/
DATA CODE(1,  4,2),CODE(2,  4,2),CODE(3,  4,2)/ 2,  5,Z0002/
DATA CODE(1,  5,2),CODE(2,  5,2),CODE(3,  5,2)/ 3,  2,Z0003/
DATA CODE(1,  6,2),CODE(2,  6,2),CODE(3,  6,2)/ 4,  7,Z0003/
DATA CODE(1,  7,2),CODE(2,  7,2),CODE(3,  7,2)/ 4,  8,Z0002/
DATA CODE(1,  8,2),CODE(2,  8,2),CODE(3,  8,2)/ 5,  9,Z0003/
DATA CODE(1,  9,2),CODE(2,  9,2),CODE(3,  9,2)/ 6, 10,Z0005/
DATA CODE(1, 10,2),CODE(2, 10,2),CODE(3, 10,2)/ 6, 11,Z0004/
DATA CODE(1, 11,2),CODE(2, 11,2),CODE(3, 11,2)/ 7, 12,Z0004/
DATA CODE(1, 12,2),CODE(2, 12,2),CODE(3, 12,2)/ 7, 13,Z0005/
DATA CODE(1, 13,2),CODE(2, 13,2),CODE(3, 13,2)/ 7, 14,Z0007/
DATA CODE(1, 14,2),CODE(2, 14,2),CODE(3, 14,2)/ 8, 15,Z0004/
DATA CODE(1, 15,2),CODE(2, 15,2),CODE(3, 15,2)/ 8, 16,Z0007/
DATA CODE(1, 16,2),CODE(2, 16,2),CODE(3, 16,2)/ 9, 17,Z0018/
DATA CODE(1, 17,2),CODE(2, 17,2),CODE(3, 17,2)/10, 18,Z0017/
DATA CODE(1, 18,2),CODE(2, 18,2),CODE(3, 18,2)/10, 19,Z0018/
DATA CODE(1, 19,2),CODE(2, 19,2),CODE(3, 19,2)/10,  1,Z0008/
DATA CODE(1, 20,2),CODE(2, 20,2),CODE(3, 20,2)/11, 21,Z0067/
DATA CODE(1, 21,2),CODE(2, 21,2),CODE(3, 21,2)/11, 22,Z0058/
DATA CODE(1, 22,2),CODE(2, 22,2),CODE(3, 22,2)/11, 23,Z006C/
DATA CODE(1, 23,2),CODE(2, 23,2),CODE(3, 23,2)/11, 24,Z0037/
```

```
DATA CODE(1, 24,2),CODE(2, 24,2),CODE(3, 24,2)/11, 25,Z0028/
DATA CODE(1, 25,2),CODE(2, 25,2),CODE(3, 25,2)/11, 26,Z0017/
DATA CODE(1, 26,2),CODE(2, 26,2),CODE(3, 26,2)/11, 27,Z0018/
DATA CODE(1, 27,2),CODE(2, 27,2),CODE(3, 27,2)/12, 28,Z00CA/
DATA CODE(1, 28,2),CODE(2, 28,2),CODE(3, 28,2)/12, 29,Z00CB/
DATA CODE(1, 29,2),CODE(2, 29,2),CODE(3, 29,2)/12, 30,Z00CC/
DATA CODE(1, 30,2),CODE(2, 30,2),CODE(3, 30,2)/12, 31,Z00CD/
DATA CODE(1, 31,2),CODE(2, 31,2),CODE(3, 31,2)/12, 32,Z0068/
DATA CODE(1, 32,2),CODE(2, 32,2),CODE(3, 32,2)/12, 33,Z0069/
DATA CODE(1, 33,2),CODE(2, 33,2),CODE(3, 33,2)/12, 34,Z006A/
DATA CODE(1, 34,2),CODE(2, 34,2),CODE(3, 34,2)/12, 35,Z006B/
DATA CODE(1, 35,2),CODE(2, 35,2),CODE(3, 35,2)/12, 36,Z00D2/
DATA CODE(1, 36,2),CODE(2, 36,2),CODE(3, 36,2)/12, 37,Z00D3/
DATA CODE(1, 37,2),CODE(2, 37,2),CODE(3, 37,2)/12, 38,Z00D4/
DATA CODE(1, 38,2),CODE(2, 38,2),CODE(3, 38,2)/12, 39,Z00D5/
DATA CODE(1, 39,2),CODE(2, 39,2),CODE(3, 39,2)/12, 40,Z00D6/
DATA CODE(1, 40,2),CODE(2, 40,2),CODE(3, 40,2)/12, 41,Z00D7/
DATA CODE(1, 41,2),CODE(2, 41,2),CODE(3, 41,2)/12, 42,Z006C/
DATA CODE(1, 42,2),CODE(2, 42,2),CODE(3, 42,2)/12, 43,Z006D/
DATA CODE(1, 43,2),CODE(2, 43,2),CODE(3, 43,2)/12, 44,Z00DA/
DATA CODE(1, 44,2),CODE(2, 44,2),CODE(3, 44,2)/12, 45,Z00DB/
DATA CODE(1, 45,2),CODE(2, 45,2),CODE(3, 45,2)/12, 46,Z0054/
DATA CODE(1, 46,2),CODE(2, 46,2),CODE(3, 46,2)/12, 47,Z0055/
DATA CODE(1, 47,2),CODE(2, 47,2),CODE(3, 47,2)/12, 48,Z0056/
DATA CODE(1, 48,2),CODE(2, 48,2),CODE(3, 48,2)/12, 49,Z0057/
DATA CODE(1, 49,2),CODE(2, 49,2),CODE(3, 49,2)/12, 50,Z0064/
DATA CODE(1, 50,2),CODE(2, 50,2),CODE(3, 50,2)/12, 51,Z0065/
DATA CODE(1, 51,2),CODE(2, 51,2),CODE(3, 51,2)/12, 52,Z0052/
DATA CODE(1, 52,2),CODE(2, 52,2),CODE(3, 52,2)/12, 53,Z0053/
DATA CODE(1, 53,2),CODE(2, 53,2),CODE(3, 53,2)/12, 54,Z0024/
DATA CODE(1, 54,2),CODE(2, 54,2),CODE(3, 54,2)/12, 55,Z0037/
DATA CODE(1, 55,2),CODE(2, 55,2),CODE(3, 55,2)/12, 56,Z0038/
DATA CODE(1, 56,2),CODE(2, 56,2),CODE(3, 56,2)/12, 57,Z0027/
DATA CODE(1, 57,2),CODE(2, 57,2),CODE(3, 57,2)/12, 58,Z0028/
DATA CODE(1, 58,2),CODE(2, 58,2),CODE(3, 58,2)/12, 59,Z0058/
DATA CODE(1, 59,2),CODE(2, 59,2),CODE(3, 59,2)/12, 60,Z0059/
DATA CODE(1, 60,2),CODE(2, 60,2),CODE(3, 60,2)/12, 61,Z002B/
DATA CODE(1, 61,2),CODE(2, 61,2),CODE(3, 61,2)/12, 62,Z002C/
DATA CODE(1, 62,2),CODE(2, 62,2),CODE(3, 62,2)/12, 63,Z005A/
DATA CODE(1, 63,2),CODE(2, 63,2),CODE(3, 63,2)/12, 64,Z0066/
DATA CODE(1, 64,2),CODE(2, 64,2),CODE(3, 64,2)/12, 66,Z0067/
DATA CODE(1, 65,2),CODE(2, 65,2),CODE(3, 65,2)/10, 20,Z000F/
DATA CODE(1, 66,2),CODE(2, 66,2),CODE(3, 66,2)/12, 67,Z00C8/
DATA CODE(1, 67,2),CODE(2, 67,2),CODE(3, 67,2)/12, 68,Z00C9/
DATA CODE(1, 68,2),CODE(2, 68,2),CODE(3, 68,2)/12, 69,Z005B/
DATA CODE(1, 69,2),CODE(2, 69,2),CODE(3, 69,2)/12, 70,Z0033/
DATA CODE(1, 70,2),CODE(2, 70,2),CODE(3, 70,2)/12, 71,Z0034/
DATA CODE(1, 71,2),CODE(2, 71,2),CODE(3, 71,2)/12, 92,Z0035/
DATA CODE(1, 72,2),CODE(2, 72,2),CODE(3, 72,2)/13, 73,Z006C/
DATA CODE(1, 73,2),CODE(2, 73,2),CODE(3, 73,2)/13, 74,Z006D/
DATA CODE(1, 74,2),CODE(2, 74,2),CODE(3, 74,2)/13, 75,Z004A/
DATA CODE(1, 75,2),CODE(2, 75,2),CODE(3, 75,2)/13, 76,Z004B/
DATA CODE(1, 76,2),CODE(2, 76,2),CODE(3, 76,2)/13, 77,Z004C/
DATA CODE(1, 77,2),CODE(2, 77,2),CODE(3, 77,2)/13, 78,Z004D/
DATA CODE(1, 78,2),CODE(2, 78,2),CODE(3, 78,2)/13, 79,Z0072/
DATA CODE(1, 79,2),CODE(2, 79,2),CODE(3, 79,2)/13, 80,Z0073/
DATA CODE(1, 80,2),CODE(2, 80,2),CODE(3, 80,2)/13, 81,Z0074/
DATA CODE(1, 81,2),CODE(2, 81,2),CODE(3, 81,2)/13, 82,Z0075/
DATA CODE(1, 82,2),CODE(2, 82,2),CODE(3, 82,2)/13, 83,Z0076/
DATA CODE(1, 83,2),CODE(2, 83,2),CODE(3, 83,2)/13, 84,Z0077/
DATA CODE(1, 84,2),CODE(2, 84,2),CODE(3, 84,2)/13, 85,Z0052/
DATA CODE(1, 85,2),CODE(2, 85,2),CODE(3, 85,2)/13, 86,Z0053/
DATA CODE(1, 86,2),CODE(2, 86,2),CODE(3, 86,2)/13, 87,Z0054/
DATA CODE(1, 87,2),CODE(2, 87,2),CODE(3, 87,2)/13, 88,Z0055/
DATA CODE(1, 88,2),CODE(2, 88,2),CODE(3, 88,2)/13, 89,Z005A/
DATA CODE(1, 89,2),CODE(2, 89,2),CODE(3, 89,2)/13, 90,Z005B/
DATA CODE(1, 90,2),CODE(2, 90,2),CODE(3, 90,2)/13, 91,Z0064/
DATA CODE(1, 91,2),CODE(2, 91,2),CODE(3, 91,2)/13, 93,Z0065/
DATA CODE(1, 92,2),CODE(2, 92,2),CODE(3, 92,2)/12, 72,Z0001/
DATA CODERD(1,  1),CODERD(2,  1),CODERD(3,  1)/ 3,  3,Z0007/
DATA CODERD(1,  2),CODERD(2,  2),CODERD(3,  2)/ 4,  4,Z000B/
DATA CODERD(1,  3),CODERD(2,  3),CODERD(3,  3)/ 3,  2,Z0006/
DATA CODERD(1,  4),CODERD(2,  4),CODERD(3,  4)/ 4,  5,Z000A/
DATA CODERD(1,  5),CODERD(2,  5),CODERD(3,  5)/ 4,  6,Z0009/
DATA CODERD(1,  6),CODERD(2,  6),CODERD(3,  6)/ 4,  7,Z0008/
DATA CODERD(1,  7),CODERD(2,  7),CODERD(3,  7)/ 5,  8,Z000F/
DATA CODERD(1,  8),CODERD(2,  8),CODERD(3,  8)/ 5,  9,Z000E/
DATA CODERD(1,  9),CODERD(2,  9),CODERD(3,  9)/ 5, 65,Z000D/
DATA CODERD(1, 10),CODERD(2, 10),CODERD(3, 10)/ 6, 11,Z0015/
DATA CODERD(1, 11),CODERD(2, 11),CODERD(3, 11)/ 6, 12,Z0014/
DATA CODERD(1, 12),CODERD(2, 12),CODERD(3, 12)/ 6, 13,Z0013/
DATA CODERD(1, 13),CODERD(2, 13),CODERD(3, 13)/ 6, 66,Z0012/
```

```
DATA CODERD(1, 14),CODERD(2, 14),CODERD(3, 14)/ 7, 15,Z0021/
DATA CODERD(1, 15),CODERD(2, 15),CODERD(3, 15)/ 7, 16,Z0020/
DATA CODERD(1, 16),CODERD(2, 16),CODERD(3, 16)/ 7, 17,Z001F/
DATA CODERD(1, 17),CODERD(2, 17),CODERD(3, 17)/ 7, 18,Z001E/
DATA CODERD(1, 18),CODERD(2, 18),CODERD(3, 18)/ 7, 19,Z001D/
DATA CODERD(1, 19),CODERD(2, 19),CODERD(3, 19)/ 7, 20,Z001C/
DATA CODERD(1, 20),CODERD(2, 20),CODERD(3, 20)/ 7, 21,Z001B/
DATA CODERD(1, 21),CODERD(2, 21),CODERD(3, 21)/ 7, 22,Z001A/
DATA CODERD(1, 22),CODERD(2, 22),CODERD(3, 22)/ 7, 23,Z0019/
DATA CODERD(1, 23),CODERD(2, 23),CODERD(3, 23)/ 7, 24,Z0018/
DATA CODERD(1, 24),CODERD(2, 24),CODERD(3, 24)/ 7, 25,Z0017/
DATA CODERD(1, 25),CODERD(2, 25),CODERD(3, 25)/ 7, 26,Z0016/
DATA CODERD(1, 26),CODERD(2, 26),CODERD(3, 26)/ 7, 27,Z0015/
DATA CODERD(1, 27),CODERD(2, 27),CODERD(3, 27)/ 7, 28,Z0014/
DATA CODERD(1, 28),CODERD(2, 28),CODERD(3, 28)/ 7, 29,Z0013/
DATA CODERD(1, 29),CODERD(2, 29),CODERD(3, 29)/ 7, 34,Z0012/
DATA CODERD(1, 30),CODERD(2, 30),CODERD(3, 30)/ 8, 31,Z0019/
DATA CODERD(1, 31),CODERD(2, 31),CODERD(3, 31)/ 8, 32,Z0018/
DATA CODERD(1, 32),CODERD(2, 32),CODERD(3, 32)/ 8, 33,Z0017/
DATA CODERD(1, 33),CODERD(2, 33),CODERD(3, 33)/ 8, 35,Z0016/
DATA CODERD(1, 34),CODERD(2, 34),CODERD(3, 34)/ 7, 36,Z0011/
DATA CODERD(1, 35),CODERD(2, 35),CODERD(3, 35)/ 8, 38,Z0015/
DATA CODERD(1, 36),CODERD(2, 36),CODERD(3, 36)/ 7, 37,Z0010/
DATA CODERD(1, 37),CODERD(2, 37),CODERD(3, 37)/ 7, 67,Z000F/
DATA CODERD(1, 38),CODERD(2, 38),CODERD(3, 38)/ 8, 39,Z0014/
DATA CODERD(1, 39),CODERD(2, 39),CODERD(3, 39)/ 8, 40,Z0013/
DATA CODERD(1, 40),CODERD(2, 40),CODERD(3, 40)/ 8, 41,Z0012/
DATA CODERD(1, 41),CODERD(2, 41),CODERD(3, 41)/ 8, 42,Z0011/
DATA CODERD(1, 42),CODERD(2, 42),CODERD(3, 42)/ 8, 43,Z0010/
DATA CODERD(1, 43),CODERD(2, 43),CODERD(3, 43)/ 8, 44,Z000F/
DATA CODERD(1, 44),CODERD(2, 44),CODERD(3, 44)/ 8, 50,Z000E/
DATA CODERD(1, 45),CODERD(2, 45),CODERD(3, 45)/ 9, 46,Z0017/
DATA CODERD(1, 46),CODERD(2, 46),CODERD(3, 46)/ 9, 47,Z0016/
DATA CODERD(1, 47),CODERD(2, 47),CODERD(3, 47)/ 9, 48,Z0015/
DATA CODERD(1, 48),CODERD(2, 48),CODERD(3, 48)/ 9, 49,Z0014/
DATA CODERD(1, 49),CODERD(2, 49),CODERD(3, 49)/ 9, 51,Z0013/
DATA CODERD(1, 50),CODERD(2, 50),CODERD(3, 50)/ 8, 68,Z000D/
DATA CODERD(1, 51),CODERD(2, 51),CODERD(3, 51)/ 9, 52,Z0012/
DATA CODERD(1, 52),CODERD(2, 52),CODERD(3, 52)/ 9, 53,Z0011/
DATA CODERD(1, 53),CODERD(2, 53),CODERD(3, 53)/ 9, 54,Z0010/
DATA CODERD(1, 54),CODERD(2, 54),CODERD(3, 54)/ 9, 55,Z000F/
DATA CODERD(1, 55),CODERD(2, 55),CODERD(3, 55)/ 9, 56,Z000E/
DATA CODERD(1, 56),CODERD(2, 56),CODERD(3, 56)/ 9, 57,Z000D/
DATA CODERD(1, 57),CODERD(2, 57),CODERD(3, 57)/ 9, 58,Z000C/
DATA CODERD(1, 58),CODERD(2, 58),CODERD(3, 58)/ 9, 59,Z000B/
DATA CODERD(1, 59),CODERD(2, 59),CODERD(3, 59)/ 9, 60,Z000A/
DATA CODERD(1, 60),CODERD(2, 60),CODERD(3, 60)/ 9, 61,Z0009/
DATA CODERD(1, 61),CODERD(2, 61),CODERD(3, 61)/ 9, 62,Z0008/
DATA CODERD(1, 62),CODERD(2, 62),CODERD(3, 62)/ 9, 63,Z0007/
DATA CODERD(1, 63),CODERD(2, 63),CODERD(3, 63)/ 9, 64,Z0006/
DATA CODERD(1, 64),CODERD(2, 64),CODERD(3, 64)/ 9, 69,Z0005/
DATA CODERD(1, 65),CODERD(2, 65),CODERD(3, 65)/ 5, 92,Z000C/
DATA CODERD(1, 66),CODERD(2, 66),CODERD(3, 66)/ 6, 14,Z0011/
DATA CODERD(1, 67),CODERD(2, 67),CODERD(3, 67)/ 7, 91,Z000E/
DATA CODERD(1, 68),CODERD(2, 68),CODERD(3, 68)/ 8, 45,Z000C/
DATA CODERD(1, 69),CODERD(2, 69),CODERD(3, 69)/ 9, 70,Z0004/
DATA CODERD(1, 70),CODERD(2, 70),CODERD(3, 70)/ 9, 89,Z0003/
DATA CODERD(1, 71),CODERD(2, 71),CODERD(3, 71)/12, 72,Z0011/
DATA CODERD(1, 72),CODERD(2, 72),CODERD(3, 72)/12, 73,Z0010/
DATA CODERD(1, 73),CODERD(2, 73),CODERD(3, 73)/12, 74,Z000F/
DATA CODERD(1, 74),CODERD(2, 74),CODERD(3, 74)/12, 75,Z000E/
DATA CODERD(1, 75),CODERD(2, 75),CODERD(3, 75)/12, 76,Z000D/
DATA CODERD(1, 76),CODERD(2, 76),CODERD(3, 76)/12, 77,Z000C/
DATA CODERD(1, 77),CODERD(2, 77),CODERD(3, 77)/12, 78,Z000B/
DATA CODERD(1, 78),CODERD(2, 78),CODERD(3, 78)/12, 79,Z000A/
DATA CODERD(1, 79),CODERD(2, 79),CODERD(3, 79)/12, 80,Z0009/
DATA CODERD(1, 80),CODERD(2, 80),CODERD(3, 80)/12, 81,Z0008/
DATA CODERD(1, 81),CODERD(2, 81),CODERD(3, 81)/12, 82,Z0007/
DATA CODERD(1, 82),CODERD(2, 82),CODERD(3, 82)/12, 83,Z0006/
DATA CODERD(1, 83),CODERD(2, 83),CODERD(3, 83)/12, 84,Z0005/
DATA CODERD(1, 84),CODERD(2, 84),CODERD(3, 84)/12, 85,Z0004/
DATA CODERD(1, 85),CODERD(2, 85),CODERD(3, 85)/12, 86,Z0003/
DATA CODERD(1, 86),CODERD(2, 86),CODERD(3, 86)/12, 87,Z0002/
DATA CODERD(1, 87),CODERD(2, 87),CODERD(3, 87)/12, 88,Z0013/
DATA CODERD(1, 88),CODERD(2, 88),CODERD(3, 88)/12, 93,Z0012/
DATA CODERD(1, 89),CODERD(2, 89),CODERD(3, 89)/11, 90,Z000B/
DATA CODERD(1, 90),CODERD(2, 90),CODERD(3, 90)/11, 71,Z000A/
DATA CODERD(1, 91),CODERD(2, 91),CODERD(3, 91)/ 7, 30,Z000D/
DATA CODERD(1, 92),CODERD(2, 92),CODERD(3, 92)/ 5, 10,Z000B/
DATA CODERD(1, 93),CODERD(2, 93),CODERD(3, 93)/12, 94,Z0001/
DATA PREDCT(1)/0/
DATA PREDCT(2)/1/
```

```
DATA PREDCT(3)/0/
DATA PREDCT(4)/1/
DATA PREDCT(5)/0/
DATA PREDCT(6)/1/
DATA PREDCT(7)/0/
DATA PREDCT(8)/1/
DATA PREDCT(9)/0/
DATA PREDCT(10)/1/
DATA PREDCT(11)/0/
DATA PREDCT(12)/1/
DATA PREDCT(13)/0/
DATA PREDCT(14)/1/
DATA PREDCT(15)/0/
DATA PREDCT(16)/1/
DATA PREDCT(17)/0/
DATA PREDCT(18)/1/
DATA PREDCT(19)/0/
DATA PREDCT(20)/1/
DATA PREDCT(21)/0/
DATA PREDCT(22)/1/
DATA PREDCT(23)/1/
DATA PREDCT(24)/1/
DATA PREDCT(25)/1/
DATA PREDCT(26)/1/
DATA PREDCT(27)/1/
DATA PREDCT(28)/1/
DATA PREDCT(29)/1/
DATA PREDCT(30)/1/
DATA PREDCT(31)/1/
DATA PREDCT(32)/1/
DATA PREDCT(33)/0/
DATA PREDCT(34)/1/
DATA PREDCT(35)/0/
DATA PREDCT(36)/1/
DATA PREDCT(37)/0/
DATA PREDCT(38)/1/
DATA PREDCT(39)/1/
DATA PREDCT(40)/1/
DATA PREDCT(41)/0/
DATA PREDCT(42)/1/
DATA PREDCT(43)/0/
DATA PREDCT(44)/1/
DATA PREDCT(45)/0/
DATA PREDCT(46)/0/
DATA PREDCT(47)/0/
DATA PREDCT(48)/1/
DATA PREDCT(49)/0/
DATA PREDCT(50)/1/
DATA PREDCT(51)/0/
DATA PREDCT(52)/1/
DATA PREDCT(53)/0/
DATA PREDCT(54)/1/
DATA PREDCT(55)/0/
DATA PREDCT(56)/1/
DATA PREDCT(57)/1/
DATA PREDCT(58)/1/
DATA PREDCT(59)/0/
DATA PREDCT(60)/1/
DATA PREDCT(61)/1/
DATA PREDCT(62)/1/
DATA PREDCT(63)/0/
DATA PREDCT(64)/1/
DATA PREDCT(65)/0/
DATA PREDCT(66)/1/
DATA PREDCT(67)/0/
DATA PREDCT(68)/0/
DATA PREDCT(69)/0/
DATA PREDCT(70)/1/
DATA PREDCT(71)/0/
DATA PREDCT(72)/1/
DATA PREDCT(73)/0/
DATA PREDCT(74)/1/
DATA PREDCT(75)/0/
DATA PREDCT(76)/1/
DATA PREDCT(77)/0/
DATA PREDCT(78)/1/
DATA PREDCT(79)/0/
DATA PREDCT(80)/1/
DATA PREDCT(81)/0/
DATA PREDCT(82)/1/
DATA PREDCT(83)/0/
DATA PREDCT(84)/1/
```

```
      DATA PREDCT(85)/0/
      DATA PREDCT(86)/1/
      DATA PREDCT(87)/1/
      DATA PREDCT(88)/1/
      DATA PREDCT(89)/0/
      DATA PREDCT(90)/1/
      DATA PREDCT(91)/1/
      DATA PREDCT(92)/1/
      DATA PREDCT(93)/0/
      DATA PREDCT(94)/1/
      DATA PREDCT(95)/1/
      DATA PREDCT(96)/1/
      DATA PREDCT(97)/0/
      DATA PREDCT(98)/1/
      DATA PREDCT(99)/0/
      DATA PREDCT(100)/0/
      DATA PREDCT(101)/0/
      DATA PREDCT(102)/1/
      DATA PREDCT(103)/0/
      DATA PREDCT(104)/0/
      DATA PREDCT(105)/0/
      DATA PREDCT(106)/1/
      DATA PREDCT(107)/0/
      DATA PREDCT(108)/1/
      DATA PREDCT(109)/0/
      DATA PREDCT(110)/1/
      DATA PREDCT(111)/0/
      DATA PREDCT(112)/1/
      DATA PREDCT(113)/0/
      DATA PREDCT(114)/1/
      DATA PREDCT(115)/0/
      DATA PREDCT(116)/1/
      DATA PREDCT(117)/0/
      DATA PREDCT(118)/1/
      DATA PREDCT(119)/0/
      DATA PREDCT(120)/1/
      DATA PREDCT(121)/0/
      DATA PREDCT(122)/1/
      DATA PREDCT(123)/0/
      DATA PREDCT(124)/1/
      DATA PREDCT(125)/0/
      DATA PREDCT(126)/1/
      DATA PREDCT(127)/0/
      DATA PREDCT(128)/1/
      E N D
      SUBROUTINE XCODLR(LENGTH,POLAR,CDELCT,CDDATA)
C
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PREBUF,PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93),PREDCT(128)
      COMMON/ERAY/ERRORS(2500)
C
C******************************* BEGIN PROGRAM ***********************
C
C    INITIALIZE MAKE UP CODE, MAKE UP CODE LENGTH
C
      MCODE=0
      MLENG=0
C
C    CHECK INPUTS
C
      IF(POLAR.LT.1.OR.POLAR.GT.2) CALL EXIT
      IF(LENGTH.LT.0.OR.LENGTH.GT.1728) CALL EXIT
C
      IF(LENGTH.LE.63) GO TO 10
C
C    CALCULATE MAKE UP CODE INDEX, CODE, LENGTH
C     AND WRITE TO CODE LINE
C
      INDEX=LENGTH/64+64
      MCODE=CODE(3,INDEX,POLAR)
      MLENG=CODE(1,INDEX,POLAR)
      CALL MI2B(MCODE,CDBUF,CDELCT+1,MLENG)
      CDELCT=CDELCT+MLENG
      CDDATA=CDDATA+MLENG
C
C    CALCULATE TERMINATING CODE INDEX, CODE, LENGTH
C     AND ADD TO CODE LINE
C
   10 CONTINUE
      INDEX=MOD(LENGTH,64)+1
```

```
      TCODE=CODE(3,INDEX,POLAR)
      TLENG=CODE(1,INDEX,POLAR)
      CALL MI28(TCODE,CDBUF,CDELCT+1,TLENG)
      CDELCT=CDELCT+TLENG
      CDDATA=CDDATA+TLENG
C
      RETURN
      END
      SUBROUTINE ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      IMPLICIT INTEGER(A-Z)
      REAL ESF
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
C*********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      DIMENSION PELBUF(60), OTBUF(60)
      COMMON/LOGIC/SEARCH,DIAG
      LOGICAL SEARCH,DIAG
C
C**************************** BEGIN PROGRAM ***************************
C
      REWIND PELFIL
      REWIND OTFIL
      ERROR=0
      OTELW=(PELMAX+32-1)/32
      OTLNCT=0
C
C   READ AN ERROR FREE LINE
C
  100 CONTINUE
      READ(PELFIL,END=600,ERR=800) INLNNO,INELCT,PELBUF
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
C
C   READ AN ERROR-CORRUPTED LINE
C
  200 CONTINUE
      READ(OTFIL,END=500,ERR=800) OTLNNO,OTELCT,OTBUF
      OTLNCT=OTLNCT+1
  300 CONTINUE
C
C   COUNT DIFFERENCES BETWEEN TRANSMITTED AND RECEIVED LINES
C
      DO 450 I=1,OTELW
      IF(OTBUF(I).EQ.PELBUF(I)) GO TO 450
      IF(.NOT.DIAG) GO TO 420
      WRITE(TERM,410) INLNNO,OTLNNO,I,PELBUF(I),OTBUF(I)
  410 FORMAT(3I8,2Z12)
  420 CONTINUE
      DO 440 J=1,32
      IF(I4B(OTBUF(I),J,1).NE.I4B(PELBUF(I),J,1)) ERROR=ERROR+1
  440 CONTINUE
  450 CONTINUE
      IF(OTLNNO-INLNNO) 200,100,580
C
C   ERROR LINE NUMBER GREATER THAN GOOD LINE NUMBER;
C   COUNT DIFFERENCES BETWEEN GOOD AND ALL WHITE LINE
C
  500 CONTINUE
      DO 550 I=1,OTELW
      IF(PELBUF(I).EQ.0) GO TO 550
      IF(.NOT.DIAG) GO TO 520
      WRITE(TERM,410) INLNNO,OTLNNO,I,PELBUF(I),OTBUF(I)
  520 CONTINUE
      DO 540 J=1,32
      IF(I4B(PELBUF(I),J,1).NE.0) ERROR=ERROR+1
  540 CONTINUE
  550 CONTINUE
C
  580 READ(PELFIL,END=600,ERR=800) INLNNO,INELCT,PELBUF
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 580
C
      GO TO 300
C
C   CALCULATE ERROR SENSITIVITY FACTOR
C
  600 CONTINUE
```

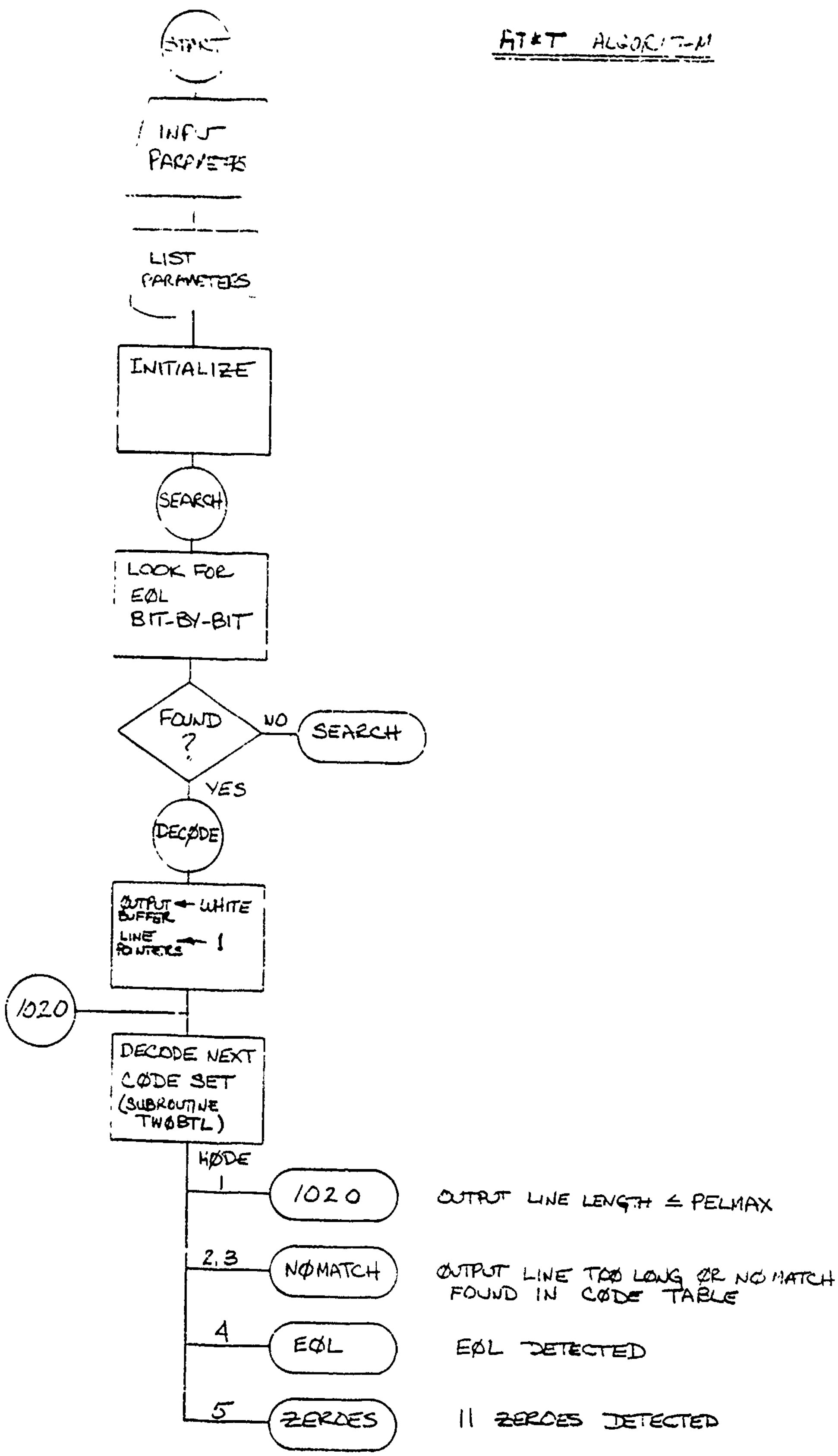
```
      ESF=0.
      IF(ERRCNT.LE.0) GO TO 650
      ESF=FLOAT(ERROR)/FLOAT(ERRCNT)
  650 CONTINUE
C
      WRITE(LPFIL,700) ERROR,ERRCNT,ESF,OTLNCT
  700 FORMAT('0NUMBER OF  INCORRECT PELS =',I10/
     *       '0NUMBER OF BITS IN ERROR TRANSMITTED =',I10/
     *       '0ERROR SENSITIVITY FACTOR =',F12.4/
     *       '0TOTAL NUMBER OF OUTPUT LINES PROCESSED = ',I8)
C
      RETURN
  800 CONTINUE
      STOP 800
      E N D
      SUBROUTINE STATS(LENGTH,INLNCT,DIAG)
      IMPLICIT INTEGER(A-Z)
C
      INTEGER MTT(5),ITT(2,5),LENGTH(INLNCT)
      REAL STT(2,5),SUM,SUMSQ
      LOGICAL DIAG
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      DATA MTT/0,24,48,96,192/
C
C*************************BEGIN PROGRAM**********************************
C
      DO 300 I=1,5
      ITT(1,I)=10000
      ITT(2,I)=0
      SUM=0.
      SUMSQ=0.
      DO 100 J=1,INLNCT
C
C   FIND FILLED LINE LENGTH
C
      LEN=MAX0(LENGTH(J),MTT(I))
      IF(DIAG) WRITE(TERM,50) LEN
   50 FORMAT(I8)
C
C   FIND MINIMUM LINE LENGTH
C
      ITT(1,I)=MIN0(LEN,ITT(1,I))
C
C   FIND MAXIMUM LINE LENGTH
C
      ITT(2,I)=MAX0(LEN,ITT(2,I))
C
C   FIND SUM OF LENGTHS
C
      SUM=SUM+FLOAT(LEN)
      SUMSQ=SUMSQ+(FLOAT(LEN))**2
  100 CONTINUE
C
C   FIND SAMPLE MEAN AND STANDARD DEVIATION
C
      STT(1,I)=SUM/FLOAT(INLNCT)
      STT(2,I)=SQRT((SUMSQ-(SUM**2)/FLOAT(INLNCT))/FLOAT(INLNCT-1))
  300 CONTINUE
C
      WRITE(LPFIL,400)(ITT(1,I),I=1,5)
  400 FORMAT(
     *'0                          MINIMUM TRANSMISSION TIME (4800 BPS)'/
     *'   CODED LINE'/
     *'     LENGTH                0 MS     5 MS    10 MS    20 MS    40 MS'/
     *'   STATISTICS:'//
     *'    MINIMUM',10X,5(I8)//)
      WRITE(LPFIL,410)(ITT(2,I),I=1,5)
  410 FORMAT(
     *'    MAXIMUM',10X,5(I8)//)
      WRITE(LPFIL,420)(STT(1,I),I=1,5)
  420 FORMAT(
     *'    SAMPLE MEAN',9X,5(F8.2)//)
      WRITE(LPFIL,430)(STT(2,I),I=1,5)
  430 FORMAT(
     *'    STANDARD DEVIATION',2X,5(F8.2))
C
      RETURN
      E N D
0          END OF DCEC UPRINT PROGRAM          LINES PRINTED=  1638
```

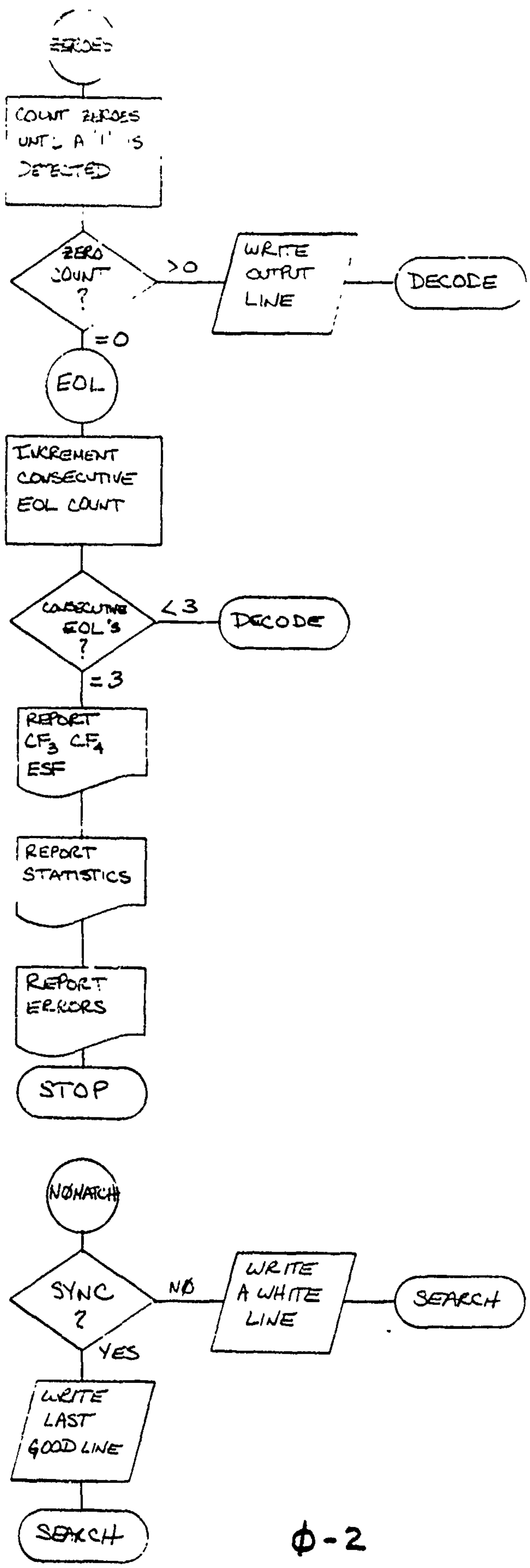
APPENDIX O

PROGRAM FLOW CHART

FOR AT&T ALGORITHM

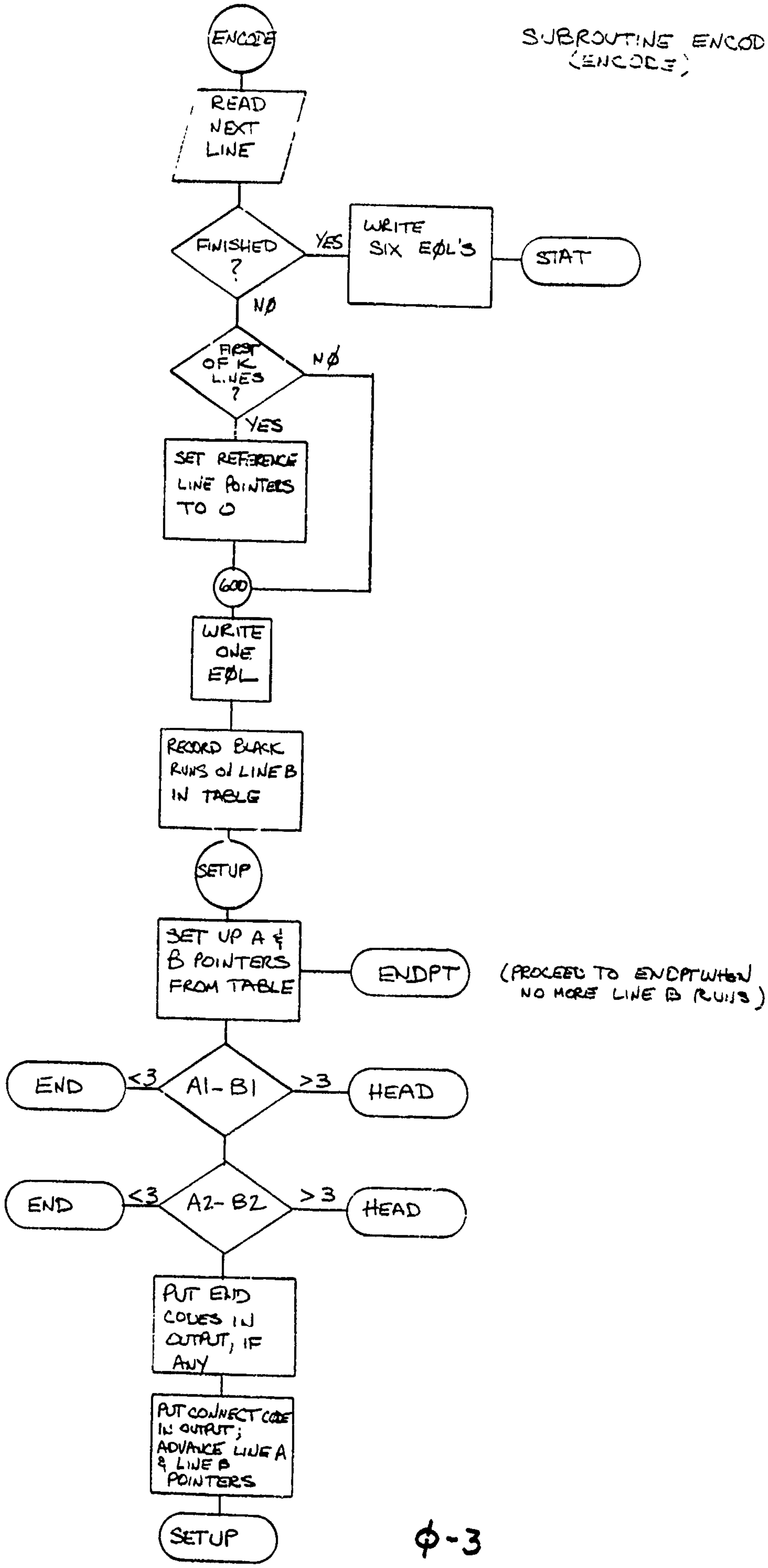
AT&T ALGORITHM

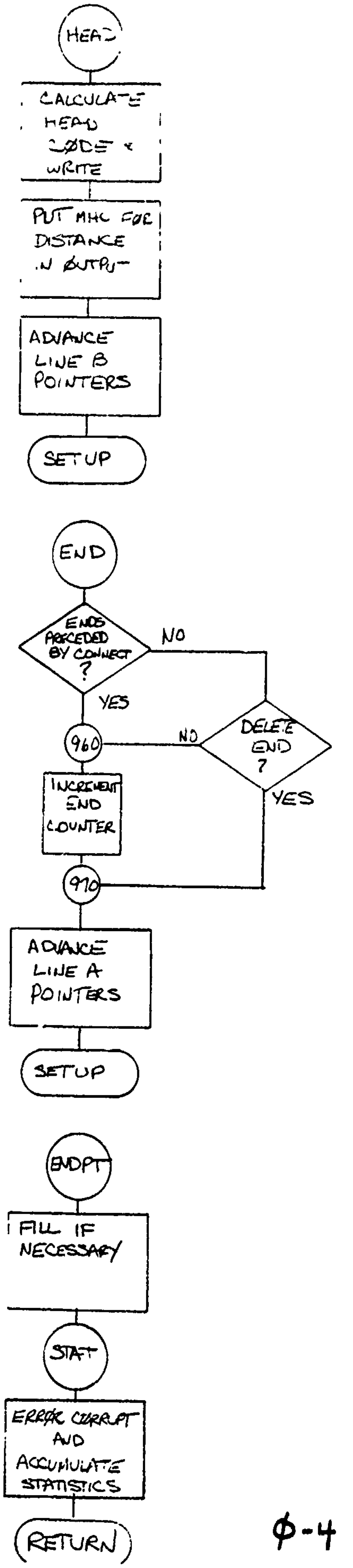



ZEROES
COUNT ZEROES UNTIL A '1' IS DETECTED
ZERO COUNT ?
>0
WRITE OUTPUT LINE
DECODE
=0
EOL
INCREMENT CONSECUTIVE EOL COUNT
CONSECUTIVE EOL'S ?
<3
DECODE
=3
REPORT CF3 CF4 ESF
REPORT STATISTICS
REPORT ERRORS
STOP
NOMATCH
SYNC ?
NO
WRITE A WHITE LINE
SEARCH
YES
WRITE LAST GOOD LINE
SEARCH

BTL 3/7

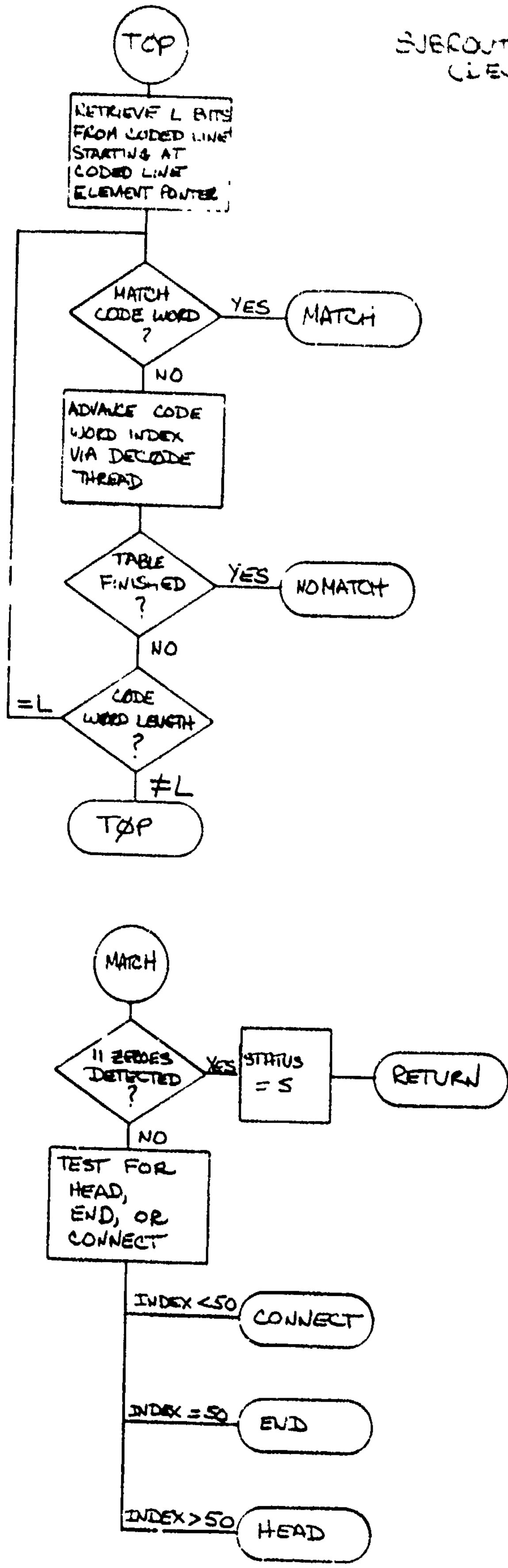
SUBROUTINE ENCODB
(ENCODE)

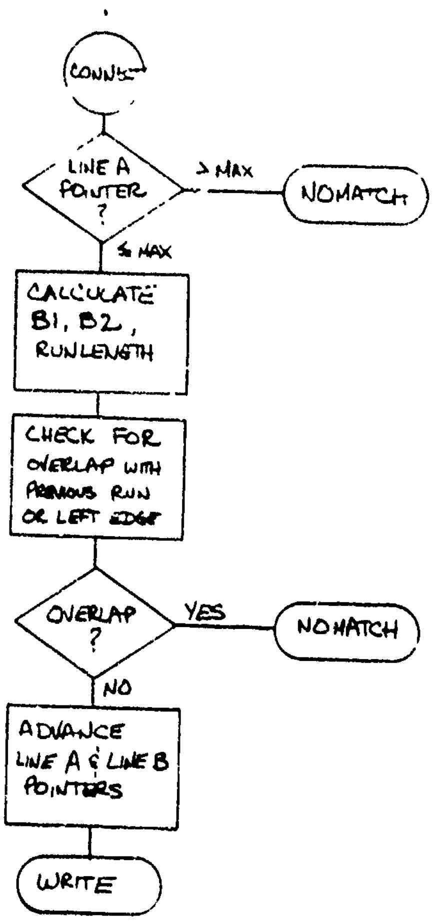


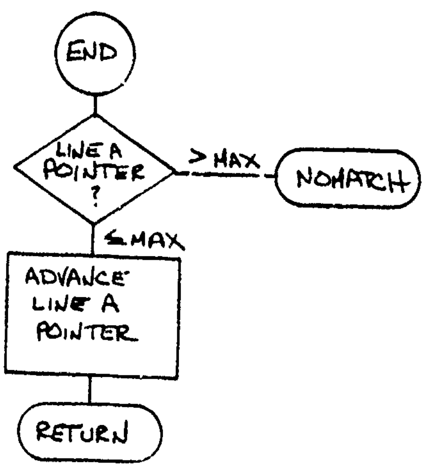

HEAD
CALCULATE HEAD CODE & WRITE
PUT MHC FOR DISTANCE IN OUTPUT
ADVANCE LINE B POINTERS
SETUP
END
ENDS PRECEDED BY CONNECT ?
NO
YES
960
NO
DELETE END ?
YES
INCREMENT END COUNTER
970
ADVANCE LINE A POINTERS
SETUP
ENDPT
FILL IF NECESSARY
STAT
ERROR CORRUPT AND ACCUMULATE STATISTICS
RETURN

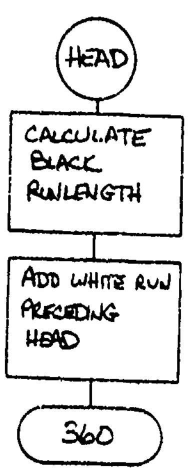
SUBROUTINE TWOBTL
(DECODE)

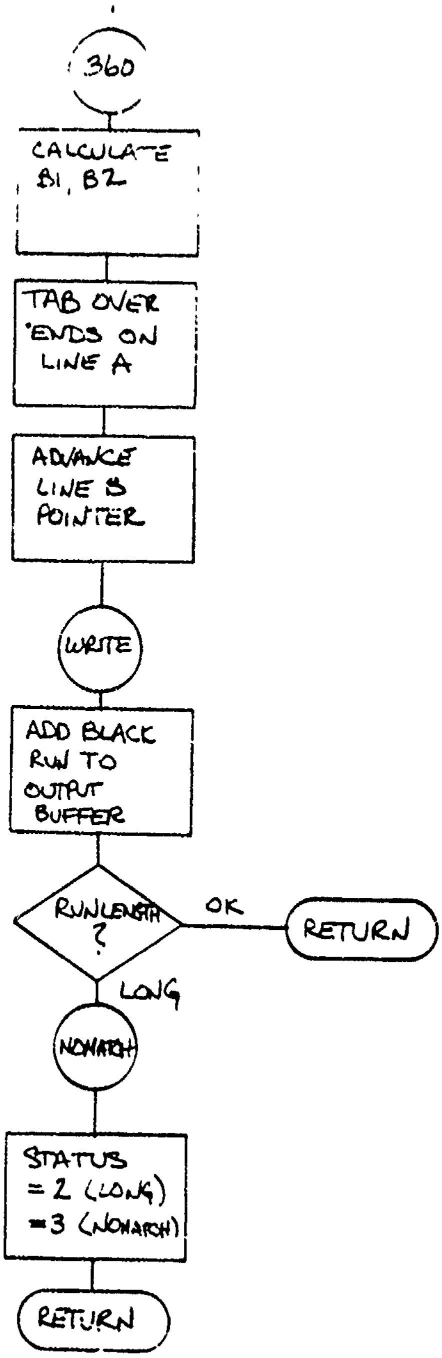
FL 5 7




CONNECT
LINE A POINTER ?
> MAX
NOMATCH
≤ MAX
CALCULATE B1, B2, RUNLENGTH
CHECK FOR OVERLAP WITH PREVIOUS RUN OR LEFT EDGE
OVERLAP ?
YES
NOMATCH
NO
ADVANCE LINE A & LINE B POINTERS
WRITE


END
LINE A POINTER ?
> MAX
NOMATCH
≤ MAX
ADVANCE LINE A POINTER
RETURN


HEAD
CALCULATE BLACK RUNLENGTH
ADD WHITE RUN PRECEDING HEAD
360



360
CALCULATE B1, B2
TAB OVER ENDS ON LINE A
ADVANCE LINE B POINTER
WRITE
ADD BLACK RUN TO OUTPUT BUFFER
RUNLENGTH ?
OK
RETURN
LONG
NOMATCH
STATUS = 2 (LONG) = 3 (NOMATCH)
RETURN

APPENDIX P

COMPUTER PROGRAM CODE LISTING

AT&T ALGORITHM

```
START OF UCSC JPRINT PROGRAM           DSNAME=D2021.UTL.FORT
C       PROGRAM UTL
        IMPLICIT INTEGER(A-Z)
        REAL CF3,CF4,ERRATE
C******** LABELLED COMMON /G32BIT/ *******
C
        COMMON /G32BIT/MASK(32),CUMASK(32),LIMIT(32),L2BIT(32)
        INTEGER MASK,CUMASK,LIMIT,L2BIT
C
        COMMON/BUFF/PELBUF(60,2),CDBUF(240),CTBUF(60,2),
      *             STFBUF(240), STAT(3000),
      *             PIN(2,864,2),POT(2,864,2),PTIMAX(2),PTOMAX(2)
        COMMON/HUFF/CODE(3,92,2),CODEND(3,93)
        COMMON/ERAY/ERRORS(2500)
C************************ FILE DEFINITIONS ********************
C
        COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************** LABELLED COMMON VARIABLES *********************
C
        COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
        COMMON/PVAR/INLNNO,CTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
      *             CDELCT,INELCT,TCDATA,TCDEL,ERFPNT,EFFCFF,ERRLIM,
      *             ERRCNT,INLNCT,CONSEC,LNNOBF,
      *             INCOD,INREF,CTCOD,CTREF,STFBIT,
      *             PTAO,PTBO
        COMMON/ICHAR/DD,II,MM,TT,NN,YY
        COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
      *CONECT
        LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C
C     READ INPUT PARAMETERS
   90 WRITE(TERM,100)
  100 FORMAT('$PARAMETERS: INPUT(=I), OR DEFAULT(=D)?')
      READ(TERM,110,ERR=90) INSW
  110 FORMAT(A1)
      IF (INSW.EQ.DD) GO TO 315
      IF (INSW.NE.II) GO TO 90
C
C     READ DIAGNOSTIC SWITCH
C
  114 WRITE(TERM,115)
  115 FORMAT('$DIAGNOSTIC PRINTOUT? (Y OR N): ')
      READ(TERM,110) INSW
      IF(INSW.EQ.YY) GO TO 116
      IF(INSW.EQ.NN) GO TO 120
      GO TO 114
  116 CONTINUE
      DIAG=.TRUE.
C
C     READ MAXIMUM NUMBER OF PELS PER LINE
C
  120 CONTINUE
      WRITE(TERM,130)
  130 FORMAT('$ENTER MAXIMUM NUMBER OF PELS PER LINE: ')
      READ(TERM,140,ERR=120) PELMAX
  140 FORMAT(I4)
      IF(PELMAX.GE.1.AND.PELMAX.LE.1728) GO TO 160
      WRITE(TERM,150) PELMAX
  150 FORMAT(' 0NUMBER OUT OF RANGE (=',I6,')')
      GO TO 120
C
C     READ VERTICAL SAMPLING
C
  160 CONTINUE
      WRITE(TERM,170)
  170 FORMAT('$ENTER VERTICAL SAMPLING: ')
      READ(TERM,180,ERR=160) VRES
  180 FORMAT(I2)
      IF(VRES.GE.1.AND.VRES.LE.10) GO TO 190
      WRITE(TERM,150) VRES
      GO TO 160
C
C     READ PARAMETER K
C
  190 CONTINUE
      WRITE(TERM,192)
  192 FORMAT('$ENTER PARAMETER K: ')
      READ(TERM,140,ERR=190) K
      IF(K.GE.1.AND.K.LE.3000) GO TO 200
      WRITE(TERM,150) K
      GO TO 190
```

```
C
C   READ ERROR PATTERN PHASE
C
  200 CONTINUE
      WRITE(TERM,210)
  210 FORMAT('$ENTER ERROR PATTERN PHASE: ')
      READ(TERM,220,ERR=200) EPHASE
  220 FORMAT(I1)
      IF(EPHASE.GE.0.AND.EPHASE.LE.3) GO TO 240
      WRITE(TERM,150) EPHASE
      GO TO 200
C
C   READ MINIMUM COMPRESSED LINE LENGTH
C
  240 CONTINUE
      WRITE(TERM,250)
  250 FORMAT('$ENTER MINIMUM COMPRESSED LINE LENGTH: ')
      READ(TERM,140,ERR=240) CMPMAX
      IF(CMPMAX.GE.0.AND.CMPMAX.LE.1728) GO TO 320
      WRITE(TERM,150) CMPMAX
      GO TO 240
C
C   READ NUMBER OF SCAN LINES TO BE PROCESSED
  320 CONTINUE
      WRITE(TERM,330)
  330 FORMAT('$NUMBER OF SCAN LINES TO BE PROCESSED=? ')
      READ(TERM,140,ERR=320) LINMAX
      IF(LINMAX.GE.1.AND.LINMAX.LE.3000) GO TO 280
      WRITE(TERM,150) LINMAX
      GO TO 320
C
C   READ ERROR MODE
C
  280 CONTINUE
      WRITE(TERM,290)
  290 FORMAT('$ERROR MODE=? (M=MANUAL,T=TAPE,N=NO ERRORS)')
      READ(TERM,110,ERR=280) ERRMOD
      IF(ERRMOD.EQ.MM) GO TO 300
      IF(ERRMOD.EQ.TT) GO TO 315
      IF(ERRMOD.NE.NN) GO TO 280
      GO TO 350
C
C   READ ERROR LOCATIONS
C
  300 CONTINUE
      ERRLIM=1
  305 READ(TERM,140) ERRORS(ERRLIM)
      IF(ERRORS(ERRLIM).EQ.9999) GO TO 310
      ERRLIM=ERRLIM+1
      GO TO 305
  310 CONTINUE
      ERRLIM=ERRLIM-1
      GO TO 350
C
C   READ ERROR TAPE FILE AND OPEN
C
  315 CONTINUE
C
      ERRLIM=1
      READ(ERFIL,318,END=317) ERRORS(ERRLIM)
      ERRLIM=ERRLIM+1
  316 READ(ERFIL,318,END=317) ERRORS(ERRLIM)
  318 FORMAT(I16)
      ERRORS(ERRLIM)=ERRORS(ERRLIM)+ERRORS(ERRLIM-1)
      ERRLIM=ERRLIM+1
      GO TO 316
  317 ERRLIM=ERRLIM-1
C
  350 CONTINUE
C
  360 CONTINUE
C   WRITE INPUT PARAMETERS
C
      WRITE(LPFIL,400) PELMAX,VRES,K,EPHASE,CMPMAX,LINMAX
  400 FORMAT('1INPUT PARAMETERS:'/
     *        '0MAXIMUM NUMBER OF PELS PER LINE=',I6/
     *        '0VERTICAL SAMPLING: N=',I4/
     *        '0PARAMETER K =',I4/
     *        '0ERROR PATTERN PHASE =',I4/
     *        '0MINIMUM COMPRESSED LINE LENGTH =',I4,' BITS'/
     *        '0NUMBER OF SCAN LINES TO BE PROCESSED =',I6)
      IF(ERRMOD.EQ.NN) WRITE(LPFIL,410)
```

```
  410 FORMAT ('0NO ERRORS INSERTED')
      IF(ERRMOD.EQ.MM) WRITE(TERM,140) (ERRORS(I),I=1,ERRLIM)
      IF(ERRMOD.EQ.TT) WRITE(TERM,420) ERRLIM
  420 FORMAT(I12,' ERRORS OBTAINED FROM ERROR TAPE')
C******************** BEGIN PROGRAM *************************************
C
C     INITIALIZE
C
      TCDEL=0
      TCDATA=0
      ERRPNT=1
      ERRCNT=0
      INLNCT=0
      ERRBFF=EPHASE*1024
      CDELCT=32
      CTELP=1
      CDELP=32+1
      CONSEC=1
      INREF=1
      INCOD=2
      OTREF=1
      OTCOD=2
      CMPMAX=MAX0(CMPMAX,13)
      PTAD=1
      PTBD=1
      PTIMAX(1)=0
      PTIMAX(2)=0
      PTOMAX(1)=0
      PTOMAX(2)=0
      STFBIT=0
C
      DO 800 I=1,240
      STFBUF(I)=0
      CDBUF(I)=0
  800 CONTINUE
      DO 850 I=1,60
      OTBUF(I,OTREF)=0
      OTBUF(I,OTCOD)=0
      PELBUF(I,INREF)=0
      PELBUF(I,INCOD)=0
  850 CONTINUE
      SEARCH=.TRUE.
      SYNC=.FALSE.
      WRITE=.FALSE.
C
C     SEARCH MODE: LOOK FOR EOL BIT-BY-BIT
C
 900  CONTINUE
      CALL GETLB(12,MODE,LBITS,L)
      GO TO (910,930,920,920),MODE
      STOP 900
 910  CONTINUE
C
C     EOL NOT FOUND; ADVANCE POINTER AND TRY AGAIN
C
      CDELP=CDELP+1
      GO TO 900
 920  CONTINUE
      STOP 920
 930  CONTINUE
C
C     EOL FOUND
C
      SEARCH=.FALSE.
      CDELP=CDELP+L
      IF(WRITE) GO TO 935
      WRITE=.TRUE.
      GO TO 960
 935  CONTINUE
C
C     SET OUTPUT DECODE LINE TO 0 AND WRITE OUT
      DO 950 I=1,60
      OTBUF(I,OTCOD)=0
 950  CONTINUE
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
      OTLNNO=LNNOBF
 960  CONTINUE
      IF(MODE-2)965,1000,900
 965  STOP 965
 1000 CONTINUE
C
C     PERFORM DECODE OF A COMPLETE LINE
```

```
C    FIRST, SET OUTPUT BUFFER TO WHITE
C    (ONLY BLACK RUNS WILL BE INSERTED)
C
       DO 1010 I=1,60
       OTBUF(I,OTCOD)=0
 1010  CONTINUE
C
       INDEX=25
       OTELP=1
       PTAD=1
       PTBD=1
C
 1020  CONTINUE
       CALL TWOBTL(INDEX,COLOR,STATUS,L)
       GO TO (1030,1070,1070,1035,1032),STATUS
C              1    2    3    4    5
       STOP 1030
C
C    RUN ADDED; CHECK LENGTH OF OUTPUT LINE
C
 1030  CONTINUE
       ONE=.TRUE.
       IF(OTELP-1-PELMAX) 1031,1031,1050
 1031  CONTINUE
       INDEX=25
       GO TO 1020
C
C    11 ZEROES DETECTED; CHECK FOR FILL AND LOOK FOR EOL
C
 1032  CONTINUE
       CDELP=CDELP+L
       ZERO=-1
 1033  CONTINUE
       ZERO=ZERO+1
       CALL GETLB(1,MODE,LBITS,L)
C
       GO TO (1034,1050,1050,1050),MODE
C
C    CHECK FOR FILL
C
 1034  CONTINUE
C
       CDELP=CDELP+L
       IF(LBITS.EQ.0) GO TO 1033
       IF(ZERO.LE.0.AND.OTELP.LE.1) GO TO 1040
C
C    EOL FOUND
C
       GO TO 1050
C
C    EOL DETECTED
C
 1035  CONTINUE
       CDELP=CDELP+L
       STATUS=4
 1040  CONTINUE
       IF(OTELP.LE.1) CONSEC=CONSEC+1
       IF(CONSEC-2)1080,1000,2000
C
C    PROBLEMS,PROBLEMS
C
 1050  STOP 1050
C
C    LINE LENGTH CORRECT, EOL DETECTED PROPERLY; WRITE OUTPUT LINE
C
 1060  CONTINUE
       WRITE(OTFIL)OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
       OTLNNO=LNNOBF
       CONSEC=1
       IF(ONE) SYNC=.TRUE.
       TEMP=OTREF
       OTREF=OTCOD
       OTCOD=TEMP
       GO TO 1000
C
C    LINE TOO LONG OR NO MATCH
C
 1070  CONTINUE
       WRITE=.FALSE.
C
C    LINE SHORT
C
```

```
 1080 CONTINUE
      IF(.NOT.SYNC) GO TO 1090
C
C    WRITE LAST GOOD LINE
C
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTREF),I=1,60)
      SYNC=.FALSE.
      GO TO 1110
 1090 CONTINUE
C
C    WRITE A WHITE LINE
C
      DO 1100 I=1, 60
 1100 OTBUF(I,OTCOD)=0
      WRITE(OTFIL) OTLNNO,PELMAX,(OTBUF(I,OTCOD),I=1,60)
 1110 OTLNNO=LNNOBF
      IF(STATUS.EQ.4) GO TO 1000
      SEARCH=.TRUE.
      GO TO 900
C
C    END OF MESSAGE
C
 2000 CONTINUE
      WRITE(LPFIL,2010) CONSEC
 2010 FORMAT('0END OF MESSAGE DETECTED (',I2,' EOL''S)')
C
C    REPORT COMPRESSION FACTOR, ERROR SENSITIVITY FACTOR,BIT ERROR RATE
C
      ERRATE=FLOAT(ERRCNT)/FLOAT(TCDEL)
      WRITE(LPFIL,2020) TCDEL,TCDATA,STFBIT,INLNCT,ERRATE
 2020 FORMAT('0TOTAL NUMBER OF CODED BITS = ',I8/
     *       '0TOTAL NUMBER OF CODED DATA BITS = ',I8/
     *       '0TOTAL NUMBER OF STUFFING BITS = ',I8/
     *       '0TOTAL NUMBER OF INPUT LINES PROCESSED = ',I8/
     *       '0BIT ERROR RATE = ',G14.6)
C
      CALL STATS(STAT,INLNCT,DIAG)
      CF3=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDEL)
      CF4=FLOAT(PELMAX)*FLOAT(INLNCT)/FLOAT(TCDATA)
C
      WRITE(LPFIL,2030) CF3,CF4
 2030 FORMAT('0COMPRESSION FACTOR FOR G3 MACHINE (CF3) =',F8.4/
     *       '0COMPRESSION FACTOR FOR G4 MACHINE (CF4) =',F8.4)
C
      CALL ERRMES(PELBUF,OTBUF,PELMAX,VRES,ERRCNT)
C
      STOP
      END
      SUBROUTINE GETLB(LBITS,MODE,WRD,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),CTBUF(60,2),
     *            STFBUF(240), STAT(2000),
     *            PIN(2,864,2),POT(2,864,2),FTIMAX(2),FTCMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93)
      COMMON/ERAY/ERRORS(2500)
C********************** LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMCD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERROFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,
     *            INCOD,INREF,OTCOD,OTREF,STFBIT,
     *            PTAO,PTBO
      COMMON/ICHAR/OO,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *CONECT
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C***************************** BEGIN PROGRAM **************************
C
      MODE=4
C
C    RETRIEVE NEXT BIT FROM CDBUF
C
  100 CONTINUE
C
C    ENCODE A NEW LINE IF NECESSARY
C
```

```
      IF(LBITS+CDELP-1.LE.CDELCT) GO TO 200
      IF(CDELCT-CDELP+1) 170,190,180
  170 STOP 170
  180 CONTINUE
      STFBUF(1)=I4B(STFBUF,CDELP,CDELCT-CDELP+1)
  190 CONTINUE
      CDELP= 32-(CDELCT-CDELP)
      CALL ENCOD )
 200  CONTINUE
      WRD=I4B(STFBUF,CDELP,LBITS)
      L=LBITS
      IF(L.EQ.11.AND.WRD.EQ.0) GO TO 400
      IF(L.EQ.12.AND.WRD.EQ.CODE(3,92,1)) GO TO 300
  250 CONTINUE
      MODE=1
      RETURN
  300 CONTINUE
      MODE=2
      RETURN
  400 CONTINUE
      MODE=3
      RETURN
      END
      SUBROUTINE ENCOD8
C
      IMPLICIT INTEGER(A-Z)
C
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),L1BIT(32),LZBIT(32)
      INTEGER MASK,COMASK,L1BIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),DTBUF(50,2),
     *            STFBUF(240), STAT(3000),
     *            PIN(2,864,2),POT(2,864,2),FTIMAX(2),FTCMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,92)
      COMMON/ERAY/ERRORS(2500)
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,CTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,CTLNNO,DTCLW,INELP,CDELP,DTCLP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,
     *            INCOD,INREF,CTCOD,CTREF,STFBIT,
     *            PTAO,PTBO
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *CONECT
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C
C*********************************** BEGIN PROGRAM *******************
C
C   INITIALIZE VARIABLES
C
      CDELCT=32
      CDDATA=0
      DO 50 I=2,240
      CDBUF(I)=0
      STFBUF(I)=0
   50 CONTINUE
C
C   READ INPUT PICTURE FILE
C
  100 CONTINUE
      READ(PELFIL,END=120,ERR=500)
     * INLNNO,INELCT,(PELBUF(I,INCOD),I=1,60)
      IF(MOD(INLNNO-1,VRES).NE.0) GO TO 100
      IF(INELCT.LT.PELMAX) CALL EXIT
      INLNCT=INLNCT+1
C
C   LOAD OUTPUT LINE NUMBER BUFFER
C
      LNNOBF=INLNNO
      IF(SEARCH)DTLNNO=LNNOBF
C
      IF(INLNNO.LE.LINMAX) GO TO 140
C
C   WRITE SIX EOL'S
```

```
C
  120 CONTINUE
      DO 130 I=1,6
      CALL M123(CODE(3,92,1),CDBUF,CDELCT+1,CODE(1,92,1))
      CDELCT=CDELCT+CODE(1,92,1)
  130 CONTINUE
      DO 135 I=1,6
      STFBUF(I)=CDBUF(I)
  135 CONTINUE
      GO TO 400
C
C     FIRST OF K LINES?
C
  140 CONTINUE
      IF(MOD(INLNCT-1,K).NE.0) GO TO 600
C
C     SET REFERENCE LINE POINTERS TO ZERO
C
      PTIMAX(INREF)=1
      PIN(1,1,INREF)=PELMAX+4
      PIN(2,1,INREF)=PELMAX+5
C
C
C     TWO-DIMENSIONAL CODING
C
  600 CONTINUE
C
C     WRITE ONE EOL
C
      CALL M123(CODE(3,92,1),CDBUF,CDELCT+1,CODE(1,92,1))
      CDELCT=CDELCT+CODE(1,92,1)
C
C     INITIALIZE
C
      ENDCNT=0
      CONECT=.TRUE.
      PTI=1
      I=1
  630 CONTINUE
      PEL=I48(PELBUF(1,INCOD),I,1)
      IF(PEL)640,650,660
  640 STOP 640
C
C     BLACK PEL NOT FOUND
C
  650 CONTINUE
      I=I+1
      IF(I-(PELMAX+1)) 630,710,710
C
C     BLACK PEL FOUND; RECORD AND LOOK FOR WHITE PEL
C
  660 CONTINUE
      PIN(1,PTI,INCOD)=I
      I=I+1
  670 CONTINUE
      PEL=I48(PELBUF(1,INCOD),I,1)
      IF(PEL)680,700,690
  680 STOP 680
C
C     WHITE PEL NOT FOUND
C
  690 CONTINUE
      I=I+1
      IF(I.GT.(PELMAX+1)) STOP 690
      GO TO 670
C
C     WHITE PEL FOUND; RECORD END OF BLACK RUN
C
  700 CONTINUE
      PIN(2,PTI,INCOD)=I
      PTI=PTI+1
      I=I+1
      IF(I-PELMAX)630,630,710
C
C     END OF LINE
C
  710 CONTINUE
      PIN(1,PTI,INCOD)=PELMAX+4
      PIN(2,PTI,INCOD)=PELMAX+5
      PTIMAX(INCOD)=PTI
      IF(.NOT.DIAG)GO TO 720
      WRITE(TERM,715)((PIN(I,J,INCOD),I=1,2),J=1,PTI)
```

```
  715 FORMAT(2I6)
  720 CONTINUE
      B2LAST=1
C
C    SET UP A AND B POINTERS FROM TABLE
C
      PTA=1
      PTB=1
  740 CONTINUE
      A1=PIN(1,PTA,INREF)
      A2=PIN(2,PTA,INREF)
  745 CONTINUE
      B1=PIN(1,PTB,INCOD)
      B2=PIN(2,PTB,INCOD)
  750 CONTINUE
      IF(B2.GT.PELMAX+1)GO TO 210
C
C    DETERMINE APPLICABLE CODE RULE
C
      IF(A1-B1.GT.3)GO TO 800
      IF(B1-A1.GT.3)GO TO 950
C
C    /B1-A1/ LESS THAN OR EQUAL TO 3; TEST /B2-A2/
C
      IF(A2-B2.GT.3)GO TO 800
      IF(B2-A2.GT.3) GO TO 950
C
C    /B1-A1/ AND/B2-A2/ LESS THAN OR EQUAL TO 3;
C
C    PROCESS CONNECT
C
  760 CONTINUE
      IF(ENDCNT.LE.0)GO TO 770
      ENDCNT=ENDCNT-1
      CALL CODBTL(50,CDELCT,CDDATA)
      GO TO 760
  770 CONTINUE
      INDEX=(B1-A1+3)*7+(B2-A2)+4
      CALL CODBTL(INDEX,CDELCT,CDDATA)
      B2LAST=B2
      CONECT=.TRUE.
C
C    ADVANCE LINE A AND LINE B POINTERS
C
      IF(PTA.LT.PTIMAX(INREF)) PTA=PTA+1
      IF(PTB.LT.PTIMAX(INCOD)) PTB=PTB+1
      GO TO 740
C
C    PROCESS HEAD
C
  800 CONTINUE
      L=B2-B1
      D=B1-B2LAST
      IF(L.GT.40)GO TO 810
      CALL CODBTL(L+50,CDELCT,CDDATA)
      GO TO 860
C
C    LENGTH GREATER THAN 40
C
  810 CONTINUE
      IF(L.GT.71)GO TO 820
      L=L-40
      CALL CODBTL(91,CDELCT,CDDATA)
      CALL MI2B(L,CDBUF,CDELCT+1,5)
      CDELCT=CDELCT+5
      CDDATA=CDDATA+5
      GO TO 840
C
C    LENGTH GREATER THAN 71
C
  820 CONTINUE
      IF(L.GT.582)GO TO 830
      L=L-71
      CALL CODBTL(92,CDELCT,CDDATA)
      CALL MI2B(L,CDBUF,CDELCT+1,9)
      CDELCT=CDELCT+9
      CDDATA=CDDATA+9
      GO TO 840
C
C    LENGTH GREATER THAN 582
C
  830 CONTINUE
```

```
      L=L-332
      CALL CODBTL(93,CDELCT,CDDATA)
      CALL MI2B(L,CDBUF,CDELCT+1,11)
      CDELCT=CDELCT+11
      CDDATA=CDDATA+11
  840 CONTINUE
      IF(I4B(CDBUF,CDELCT-3,4).NE.0)GO TO 850
      CALL MI2B(1,CDBUF,CDELCT+1,1)
      CDELCT=CDELCT+1
      CDDATA=CDDATA+1
      STFBIT=STFBIT+1
  850 CONTINUE
      CALL CDDELN(0,1,CDELCT,CDDATA)
      B2LAST=32
      ENDCNT=0
      CONECT=.FALSE.
C
C     ADVANCE LINE B POINTERS
C
      IF(PTB.LT.PTIMAX(INCOD))PTB=PTB+1
      GO TO 745
C
C     PROCESS END
C
  950 CONTINUE
      IF(CONECT)GO TO 960
      IF(A1-B2LAST+2)970,960,960
  960 CONTINUE
      ENDCNT=ENDCNT+1
  970 CONTINUE
C
C     ADVANCE LINE A POINTERS
C
      IF(PTA.LT.PTIMAX(INREF))PTA=PTA+1
      A1=PIN(1,PTA,INREF)
      A2=PIN(2,PTA,INREF)
      GO TO 750
  210 CONTINUE
C
C       SWITCH CODE & REFERENCE LINES
C
      TEMP=INREF
      INREF=INCOD
      INCOD=TEMP
C
C     TRANSFER CDBUF TO STFBUF
C
      CDELW=(CDELCT+32-1)/32
      DO 240 I=2,CDELW
      STFBUF(I)=CDBUF(I)
  240 CONTINUE
C
C     SAVE LINE LENGTH(DATA BITS PLUS EOL)
C
      STAT(INLNCT)=MAX0(CDDATA,1)+CODE(1,92,1)
C
C     CHECK CODED LINE LENGTH
C
      FILL=CMPMAX-(CDELCT-32)
      IF(FILL) 400,400,250
C
C     CODE LINE TOO SHORT; FILL IT TO CMPMAX
  250 CONTINUE
      CDELCT=CDELCT+FILL
C
C     ACCUMULATE STATISTICS AND ERROR CORRUPT
C
  400 CONTINUE
      IF(ERRMOD.EQ.NN) GO TO 390
C
C     ERROR CORRUPT
C
  350 CONTINUE
      ERRBIT=ERRORS(ERRPNT)-ERROFF-TCDEL
      IF(ERRBIT.LE.0) GO TO 360
      IF(ERRBIT.GT.CDELCT-32) GO TO 390
C
C     ERROR IN RANGE OF CODED LINE; CHANGE APPROPRIATE BIT
C
      BIT=I4B(STFBUF,ERRBIT+32,1)
      BIT=MOD(BIT+1,2)
      CALL MI2B(BIT,STFBUF,ERRBIT+32,1)
```

```
      ERRCNT=ERRCNT+1
C
C     INCREMENT ERROR LIST POINTER
C
  360 CONTINUE
      ERRPNT=ERRPNT+1
      IF(ERRPNT.LE.ERRLIM) GO TO 250
C
C     ERROR LIST EXHAUSTED
C
      ERRPNT=ERRPNT-1
      WRITE(LPFIL,370) ERRPNT,ERRORS(ERRPNT)
  370 FORMAT('0ERROR LIST EXHAUSTED AT',I10,'TH ERROR;'/
     *       ' LAST ERROR OCCURRED AT',I10,' BITS')
      ERRMOD=NN
C
C     COMPUTE STATISTICS
C
  390 CONTINUE
      TCDEL=TCDEL+CDELCT-32
      TCDATA=TCDATA+CDDATA
      IF(DIAG) WRITE(TERM,160) INLNCT, CDDATA
  160 FORMAT(4I8)
C
      IF (.NOT.DIAG) GO TO 460
      CDELW=(CDELCT+32-1)/32
      WRITE(LPFIL,450) (CDBUF(I),I=1,CDELW)
      WRITE(LPFIL,450) (STFBUF(I),I=1,CDELW)
  450 FORMAT(6Z12)
  460 CONTINUE
      RETURN
C
  500 CONTINUE
      CALL EXIT
C
      END
      SUBROUTINE CODBTL(MODE,CDELCT,CDDATA)
      IMPLICIT INTEGER(A-Z)
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000),
     *            PIN(2,864,2),POT(2,864,2),PTIMAX(2),PTOMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93)
      COMMON/ERAY/ERRORS(2500)
C
C****************************** BEGIN PROGRAM ****************************
C
      CALL MI2B(CODERD(3,MODE),CDBUF,CDELCT+1,CODERD(1,MODE))
      CDELCT=CDELCT+CODERD(1,MODE)
      CDDATA=CDDATA+CODERD(1,MODE)
      RETURN
      END
      SUBROUTINE ONEBTL(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240), STAT(3000),
     *            PIN(2,864,2),POT(2,864,2),PTIMAX(2),PTOMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93)
      COMMON/ERAY/ERRORS(2500)
C************************ FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
C********************* LABELLED COMMON VARIABLES **********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,
     *            INCOD,INREF,OTCOD,OTREF,STFBIT,
     *            PTAO,PTBO
      COMMON/ICHAR/OD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *CONECT
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C
C****************************** BEGIN PROGRAM ****************************
C
```

```
C
C     BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODE(1,INDEX,COLOR)
      CALL  GETLB(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z9,I6)
      GO TO  (1040,1200,1205,1190),  MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODE(3,INDEX,COLOR)) GO TO 1100
C
C     NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODE( 2,INDEX,COLOR)
      IF(INDEX.GE.93) GO TO 1190
      IF(CODE(1,INDEX,COLOR).EQ.LENBIT) GO TO 1040
C
C     CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C     MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C     NOT AN EOL
C
C
C
C     TEST FOR MAKE UP OR TERMINATING CODE
C
      RUNLEN=INDEX-1
      IF(INDEX.GE.65) RUNLEN=(INDEX-64)*64
C     IF(RUNLEN.EQ.0.AND.OTELP.GT.1) GO TO 1190
      IF(RUNLEN.EQ.0) GO TO 1160
      IF(COLOR.EQ.1) GO TO 1155
      IF(RUNLEN.LT.0) STOP 1100
C
C     ADD BLACK RUN TO OUTPUT BUFFER
C
      DO 1150 I=1,RUNLEN
      CALL  MI2B(COLOR-1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
      GO TO 1160
C
C     ADD WHITE RUN TO OUTPUT BUFFER (BY DEFAULT)
C
 1155 CONTINUE
      OTELP=OTELP+RUNLEN
      IF(OTELP-1.GT.PELMAX) GO TO 1180
C
C OUTPUT LINE LESS THAN OR EQUAL TO MAX SPECIFIED
C
 1160 CONTINUE
      IF(INDEX.LT.65) GO TO 1170
      INDEX=3
      GO TO 1000
C
C     RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1170 CONTINUE
      STATUS=1
      RETURN
C
C     RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C     NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      IF(DIAG)WRITE(TERM,1195) INLNCT,INDEX,RUNLEN,LBITS,PTAO,PTBO
 1195 FORMAT('ONE',6I8)
      STATUS=3
```

```
      RETURN
C
C    EOL DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C    11 ZEROES (FIRST PART OF EOL) DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      E N D
      SUBROUTINE TWOBTL(INDEX,COLOR,STATUS,L)
      IMPLICIT INTEGER(A-Z)
C******* LABELED COMMON /G32BIT/ *******
C
      COMMON /G32BIT/MASK(32),COMASK(32),LIBIT(32),LZBIT(32)
      INTEGER MASK,COMASK,LIBIT,LZBIT
C
      COMMON/BUFF/PEL BUF(60,2),COBUF(240),OTBUF(60,2),
     *                STFBUF(240), STAT(3000),
     *                PIN(2,864,2),POT(2,864,2),PTIMAX(2),PTOMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERD(3,93)
      COMMON/ERAY/ERRORS(2500)
C********************** FILE DEFINITIONS *******************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,CTFIL,ERFIL
C
C******************** LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *                CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *                ERRCNT,INLNCT,CONSEC,LNNOBF,
     *                INCOD,INREF,OTCOD,OTREF,STFBIT,
     *                PTAO,PTBO
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *CONECT
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C
C    BEGIN DECODE LOOP; RETRIEVE NEXT CODE WORD LENGTH (L)
C
 1000 CONTINUE
 1002 LENBIT=CODERD(1,INDEX)
      CALL GETLB(LENBIT,MODE,LBITS,L)
      IF(DIAG) WRITE(TERM,1003) LENBIT,MODE,LBITS,L
 1003 FORMAT(2I6,Z12,I6)
      GO TO (1040,1200,1205,1190), MODE
      STOP 1040
 1040 CONTINUE
      IF(LBITS.EQ.CODERD(3,INDEX)) GO TO 1100
C
C    NO MATCH; ADVANCE CODE WORD INDEX VIA DECODE THREAD
C
      INDEX=CODERD(2,INDEX)
      IF(INDEX.GE.94) GO TO 1190
      IF(CODERD(1,INDEX).EQ.LENBIT) GO TO 1040
C
C    CODE WORD LONGER; FROM THE TOP
C
      GO TO 1002
C
C    MATCH FOUND
C
 1100 CONTINUE
      CDELP=CDELP+L
C
C    NOT AN EOL
C
C
C    TEST FOR CONNECT, END, HEAD CODE
C
      IF(INDEX-50) 100,200,300
C
C    CONNECT CODE EXTRACTED FROM CODE LINE
C
  100 CONTINUE
C
C    TEST LINE A POINTER
```

```
C
      IF(PTA0.GT.PTOMAX(OTREF)) GO TO 1190
C-B1
      POT(1,PTB0,OTCOD)=POT(1,PTA0,OTREF)+(INDEX-1)/7-3
C
C   CHECK FOR OVERLAP WITH THE PREVIOUS RUN
C
      IF(POT(1,PTB0,OTCOD).LE.CTELP.AND.CTELP.GT.1) GO TO 1190
C-B2
      POT(2,PTB0,OTCOD)=POT(2,PTA0,OTREF)+MOD((INDEX-1),7)-3
C
      OTELP=POT(1,PTB0,OTCOD)
      IF(OTELP.LT.1) GO TO 1190
      IF(OTELP.GT.PELMAX)GO TO 1180
      RUNLEN=POT(2,PTB0,OTCOD)-POT(1,PTB0,OTCOD)
C
C   ADVANCE LINE A AND LINE B POINTERS
C
      PTA0=PTA0+1
      PTB0=PTB0+1
      GO TO 1145
C
C  END CODE DETECTED
C
  200 CONTINUE
C
C   TEST LINE A POINTER
C
      IF(PTA0.GT.PTOMAX(OTREF)) GO TO 1190
C
C   ADVANCE LINE A POINTER
C
      PTA0=PTA0+1
      GO TO 1160
C
C   HEAD CODE DETECTED
C
  300 CONTINUE
C
C   FIRST CALCULATE BLACK RUNLENGTH
C
      IF(INDEX.GT.90) GO TO 310
      RUNLEN=INDEX-50
      GO TO 350
  310 CONTINUE
      IF(INDEX.GT.91)GO TO 320
      CALL GETLB(5,MODE,LBITS,L)
      IF(MODE.NE.1)STOP 310
      RUNLEN=LBITS+40
      GO TO 340
  320 CONTINUE
      IF(INDEX.GT.92)GO TO 330
      CALL GETLB(9,MODE,LBITS,L)
      IF(MODE.NE.1)STOP 320
      RUNLEN=LBITS+71
      GO TO 340
  330 CONTINUE
      CALL GETLB(11,MODE,LBITS,L)
      IF(MODE.NE.1)GO TO 1190
      RUNLEN=LBITS+582
  340 CONTINUE
      CDELP=CDELP+L
      IF(I4B(LBITS,32-3,4).NE.0)GO TO 350
      CALL GETLB(1,MODE,LBITS,L)
      IF(MODE.NE.1)STOP 340
      CDELP=CDELP+L
      IF(LBITS.NE.1)GO TO 1190
C
C   ADD WHITE RUN PRECEDING HEAD LENGTH
C
  350 CONTINUE
      INDEX=3
      COLOR=1
      CALL DNTBTL(INDEX,COLOR,STATE,L)
      GO TO (360,1180,1190,1200,1205),STATE
  360 CONTINUE
C-B1
      POT(1,PTB0,OTCOD)=OTELP
C-B2
      POT(2,PTB0,OTCOD)=OTELP+RUNLEN
C
C   TAB OVER ENDS ON LINE A
```

```
C
  370 CONTINUE
      IF(PTA0.GT.PTOMAX(OTREF))GO TO 400
      IF(POT(1,PTA0,OTREF).GT.POT(2,PTB0,OTCOD)-3)GO TO 400
      PTA0=PTA0+1
      GO TO 370
  400 CONTINUE
C
C   ADVANCE LINE B POINTER
C
      PTB0=PTB0+1
C
C   ADD BLACK RUN TO OUTPUT BUFFER
C
 1145 CONTINUE
      PTOMAX(OTCOD)=PTB0-1
      IF(RUNLEN)1190,1190,1147
 1147 CONTINUE
      DO 1150 I=1,RUNLEN
      CALL MI23(1,OTBUF(1,OTCOD),OTELP,1)
      OTELP=OTELP+1
      IF(OTELP-1.GT.PELMAX) GO TO 1180
 1150 CONTINUE
C
C   RUN ADDED TO OUTPUT LINE; LENGTH LESS THAN OR EQUAL TO PELMAX (1)
C
 1160 CONTINUE
      STATUS=1
      RETURN
C
C   RUN ADDED UNTIL PELMAX EXCEEDED; LINE TOO LONG (2)
C
 1180 CONTINUE
      IF(DIAG) WRITE(TERM,1185) (OTBUF(I,OTCOD),I=1,60)
 1185 FORMAT(6Z10)
      STATUS=2
      RETURN
C
C   NO MATCH FOUND IN CODE TABLE (3)
C
 1190 CONTINUE
      IF(DIAG)WRITE(TERM,1195) INLNCT,INDEX,RUNLEN,LBITS,PTA0,PTB0
 1195 FORMAT('TWO',6I8)
      STATUS=3
      RETURN
C
C   EOL DETECTED (4)
C
 1200 CONTINUE
      STATUS=4
      RETURN
C
C    11 ZEROES (FIRST PART OF EOL) DETECTED (5)
C
 1205 CONTINUE
      STATUS=5
      RETURN
      END

      BLOCK DATA
C
      IMPLICIT INTEGER(A-Z)
C*********************** FILE DEFINITIONS ********************
C
      COMMON/FILES/TERM,LPFIL,PELFIL,OTFIL,ERFIL
C
      COMMON/BUFF/PELBUF(60,2),CDBUF(240),OTBUF(60,2),
     *            STFBUF(240),  STAT(3000),
     *            PIN(2,864,2),POT(2,864,2),PTIMAX(2),PTOMAX(2)
      COMMON/HUFF/CODE(3,92,2),CODERO(3,93)
      COMMON/ERAY/ERRORS(2500)
C********************* LABELLED COMMON VARIABLES *********************
C
      COMMON/IVAR/PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX,K
      COMMON/PVAR/INLNNO,OTLNNO,OTELW,INELP,CDELP,OTELP,CDELW,
     *            CDELCT,INELCT,TCDATA,TCDEL,ERRPNT,ERRCFF,ERRLIM,
     *            ERRCNT,INLNCT,CONSEC,LNNOBF,
     *            INCOD,INREF,OTCOD,OTREF,STFBIT,
     *            PTA0,PTB0
      COMMON/ICHAR/DD,II,MM,TT,NN,YY
      COMMON/LOGIC/SEARCH,DIAG,SYNC,LSS,WRITE,ZERO,LEFT,CHCOL,ONE,
     *CONECT
```

```
      LOGICAL SEARCH,DIAG,SYNC,WRITE,LEFT,CHCOL,ONE,CONECT
C
      DATA TERM,LPFIL,PELFIL,CTFIL,EFFIL/5,6,1,2,3/
      DATA DD,II,MM,TT,NN,YY/'D','I','M','T','N','Y'/
      DATA PELMAX,VRES,EPHASE,CMPMAX,ERRMOD,LINMAX/1728,2,0,96,'T',3000/
      DATA K/2/
      DATA DIAG/.FALSE./
       DATA CODE(1,  1,1),CODE(2,  1,1),CODE(3,  1,1)/ 8, 70,Z0035/
       DATA CODE(1,  2,1),CODE(2,  2,1),CODE(3,  2,1)/ 6, 90,Z0007/
       DATA CODE(1,  3,1),CODE(2,  3,1),CODE(3,  3,1)/ 4,  4,Z0007/
       DATA CODE(1,  4,1),CODE(2,  4,1),CODE(3,  4,1)/ 4,  5,Z0008/
       DATA CODE(1,  5,1),CODE(2,  5,1),CODE(3,  5,1)/ 4,  6,Z000D/
       DATA CODE(1,  6,1),CODE(2,  6,1),CODE(3,  6,1)/ 4,  7,Z000C/
       DATA CODE(1,  7,1),CODE(2,  7,1),CODE(3,  7,1)/ 4,  8,Z000E/
       DATA CODE(1,  8,1),CODE(2,  8,1),CODE(3,  8,1)/ 4,  9,Z000F/
       DATA CODE(1,  9,1),CODE(2,  9,1),CODE(3,  9,1)/ 5, 10,Z0013/
       DATA CODE(1, 10,1),CODE(2, 10,1),CODE(3, 10,1)/ 5, 11,Z0014/
       DATA CODE(1, 11,1),CODE(2, 11,1),CODE(3, 11,1)/ 5, 12,Z0007/
       DATA CODE(1, 12,1),CODE(2, 12,1),CODE(3, 12,1)/ 5, 65,Z0006/
       DATA CODE(1, 13,1),CODE(2, 13,1),CODE(3, 13,1)/ 6, 14,Z0008/
       DATA CODE(1, 14,1),CODE(2, 14,1),CODE(3, 14,1)/ 6, 15,Z0003/
       DATA CODE(1, 15,1),CODE(2, 15,1),CODE(3, 15,1)/ 6, 16,Z0034/
       DATA CODE(1, 16,1),CODE(2, 16,1),CODE(3, 16,1)/ 6, 17,Z0035/
       DATA CODE(1, 17,1),CODE(2, 17,1),CODE(3, 17,1)/ 6, 18,Z002A/
       DATA CODE(1, 18,1),CODE(2, 18,1),CODE(3, 18,1)/ 6, 19,Z002B/
       DATA CODE(1, 19,1),CODE(2, 19,1),CODE(3, 19,1)/ 7, 20,Z0027/
       DATA CODE(1, 20,1),CODE(2, 20,1),CODE(3, 20,1)/ 7, 21,Z000C/
       DATA CODE(1, 21,1),CODE(2, 21,1),CODE(3, 21,1)/ 7, 22,Z0008/
       DATA CODE(1, 22,1),CODE(2, 22,1),CODE(3, 22,1)/ 7, 23,Z0017/
       DATA CODE(1, 23,1),CODE(2, 23,1),CODE(3, 23,1)/ 7, 24,Z0003/
       DATA CODE(1, 24,1),CODE(2, 24,1),CODE(3, 24,1)/ 7, 25,Z0004/
       DATA CODE(1, 25,1),CODE(2, 25,1),CODE(3, 25,1)/ 7, 26,Z0028/
       DATA CODE(1, 26,1),CODE(2, 26,1),CODE(3, 26,1)/ 7, 27,Z002B/
       DATA CODE(1, 27,1),CODE(2, 27,1),CODE(3, 27,1)/ 7, 28,Z0013/
       DATA CODE(1, 28,1),CODE(2, 28,1),CODE(3, 28,1)/ 7, 29,Z0024/
       DATA CODE(1, 29,1),CODE(2, 29,1),CODE(3, 29,1)/ 7, 68,Z0018/
       DATA CODE(1, 30,1),CODE(2, 30,1),CODE(3, 30,1)/ 8, 31,Z0002/
       DATA CODE(1, 31,1),CODE(2, 31,1),CODE(3, 31,1)/ 8, 32,Z0003/
       DATA CODE(1, 32,1),CODE(2, 32,1),CODE(3, 32,1)/ 8, 33,Z001A/
       DATA CODE(1, 33,1),CODE(2, 33,1),CODE(3, 33,1)/ 8, 34,Z001B/
       DATA CODE(1, 34,1),CODE(2, 34,1),CODE(3, 34,1)/ 8, 35,Z0012/
       DATA CODE(1, 35,1),CODE(2, 35,1),CODE(3, 35,1)/ 8, 35,Z0013/
       DATA CODE(1, 36,1),CODE(2, 36,1),CODE(3, 36,1)/ 8, 37,Z0014/
       DATA CODE(1, 37,1),CODE(2, 37,1),CODE(3, 37,1)/ 8, 38,Z0015/
       DATA CODE(1, 38,1),CODE(2, 38,1),CODE(3, 38,1)/ 8, 39,Z0016/
       DATA CODE(1, 39,1),CODE(2, 39,1),CODE(3, 39,1)/ 8, 40,Z0017/
       DATA CODE(1, 40,1),CODE(2, 40,1),CODE(3, 40,1)/ 8, 41,Z0028/
       DATA CODE(1, 41,1),CODE(2, 41,1),CODE(3, 41,1)/ 8, 42,Z0029/
       DATA CODE(1, 42,1),CODE(2, 42,1),CODE(3, 42,1)/ 8, 43,Z002A/
       DATA CODE(1, 43,1),CODE(2, 43,1),CODE(3, 43,1)/ 8, 44,Z002B/
       DATA CODE(1, 44,1),CODE(2, 44,1),CODE(3, 44,1)/ 8, 45,Z002C/
       DATA CODE(1, 45,1),CODE(2, 45,1),CODE(3, 45,1)/ 8, 46,Z002D/
       DATA CODE(1, 46,1),CODE(2, 46,1),CODE(3, 46,1)/ 8, 47,Z0004/
       DATA CODE(1, 47,1),CODE(2, 47,1),CODE(3, 47,1)/ 8, 48,Z0005/
       DATA CODE(1, 48,1),CODE(2, 48,1),CODE(3, 48,1)/ 8, 49,Z000A/
       DATA CODE(1, 49,1),CODE(2, 49,1),CODE(3, 49,1)/ 8, 50,Z000B/
       DATA CODE(1, 50,1),CODE(2, 50,1),CODE(3, 50,1)/ 8, 51,Z0052/
       DATA CODE(1, 51,1),CODE(2, 51,1),CODE(3, 51,1)/ 8, 52,Z0053/
       DATA CODE(1, 52,1),CODE(2, 52,1),CODE(3, 52,1)/ 8, 53,Z0054/
       DATA CODE(1, 53,1),CODE(2, 53,1),CODE(3, 53,1)/ 8, 54,Z0055/
       DATA CODE(1, 54,1),CODE(2, 54,1),CODE(3, 54,1)/ 8, 55,Z0024/
       DATA CODE(1, 55,1),CODE(2, 55,1),CODE(3, 55,1)/ 8, 56,Z0025/
       DATA CODE(1, 56,1),CODE(2, 56,1),CODE(3, 56,1)/ 8, 57,Z0058/
       DATA CODE(1, 57,1),CODE(2, 57,1),CODE(3, 57,1)/ 8, 58,Z0059/
       DATA CODE(1, 58,1),CODE(2, 58,1),CODE(3, 58,1)/ 8, 59,Z005A/
       DATA CODE(1, 59,1),CODE(2, 59,1),CODE(3, 59,1)/ 8, 60,Z005B/
       DATA CODE(1, 60,1),CODE(2, 60,1),CODE(3, 60,1)/ 8, 61,Z004A/
       DATA CODE(1, 61,1),CODE(2, 61,1),CODE(3, 61,1)/ 8, 62,Z004B/
       DATA CODE(1, 62,1),CODE(2, 62,1),CODE(3, 62,1)/ 8, 63,Z0032/
       DATA CODE(1, 63,1),CODE(2, 63,1),CODE(3, 63,1)/ 8, 64,Z0033/
       DATA CODE(1, 64,1),CODE(2, 64,1),CODE(3, 64,1)/ 8, 69,Z0034/
       DATA CODE(1, 65,1),CODE(2, 65,1),CODE(3, 65,1)/ 5, 66,Z001B/
       DATA CODE(1, 66,1),CODE(2, 66,1),CODE(3, 66,1)/ 5, 67,Z0012/
       DATA CODE(1, 67,1),CODE(2, 67,1),CODE(3, 67,1)/ 6,  2,Z0017/
       DATA CODE(1, 68,1),CODE(2, 68,1),CODE(3, 68,1)/ 7, 30,Z0037/
       DATA CODE(1, 69,1),CODE(2, 69,1),CODE(3, 69,1)/ 8,  1,Z0036/
       DATA CODE(1, 70,1),CODE(2, 70,1),CODE(3, 70,1)/ 8, 71,Z0037/
       DATA CODE(1, 71,1),CODE(2, 71,1),CODE(3, 71,1)/ 8, 72,Z0064/
       DATA CODE(1, 72,1),CODE(2, 72,1),CODE(3, 72,1)/ 8, 73,Z0065/
       DATA CODE(1, 73,1),CODE(2, 73,1),CODE(3, 73,1)/ 8, 74,Z0068/
       DATA CODE(1, 74,1),CODE(2, 74,1),CODE(3, 74,1)/ 8, 75,Z0067/
       DATA CODE(1, 75,1),CODE(2, 75,1),CODE(3, 75,1)/ 9, 76,Z00CC/
```

```
DATA CODE(1, 76,1),CODE(2, 76,1),CODE(3, 76,1)/ 9, 77,Z00CD/
DATA CODE(1, 77,1),CODE(2, 77,1),CODE(3, 77,1)/ 9, 78,Z00D2/
DATA CODE(1, 78,1),CODE(2, 78,1),CODE(3, 78,1)/ 9, 79,Z00D3/
DATA CODE(1, 79,1),CODE(2, 79,1),CODE(3, 79,1)/ 9, 80,Z00D4/
DATA CODE(1, 80,1),CODE(2, 80,1),CODE(3, 80,1)/ 9, 81,Z00D5/
DATA CODE(1, 81,1),CODE(2, 81,1),CODE(3, 81,1)/ 9, 82,Z00D6/
DATA CODE(1, 82,1),CODE(2, 82,1),CODE(3, 82,1)/ 9, 83,Z00D7/
DATA CODE(1, 83,1),CODE(2, 83,1),CODE(3, 83,1)/ 9, 84,Z00D8/
DATA CODE(1, 84,1),CODE(2, 84,1),CODE(3, 84,1)/ 9, 85,Z00D9/
DATA CODE(1, 85,1),CODE(2, 85,1),CODE(3, 85,1)/ 9, 86,Z00DA/
DATA CODE(1, 86,1),CODE(2, 86,1),CODE(3, 86,1)/ 9, 87,Z00DB/
DATA CODE(1, 87,1),CODE(2, 87,1),CODE(3, 87,1)/ 9, 88,Z009B/
DATA CODE(1, 88,1),CODE(2, 88,1),CODE(3, 88,1)/ 9, 89,Z0099/
DATA CODE(1, 89,1),CODE(2, 89,1),CODE(3, 89,1)/ 9, 91,Z009A/
DATA CODE(1, 90,1),CODE(2, 90,1),CODE(3, 90,1)/ 6, 13,Z0018/
DATA CODE(1, 91,1),CODE(2, 91,1),CODE(3, 91,1)/ 9, 92,Z009B/
DATA CODE(1, 92,1),CODE(2, 92,1),CODE(3, 92,1)/12, 93,Z0001/
DATA CODERD(1,  1),CODERD(2,  1),CODERD(3,  1)/11, 49,Z0609/
DATA CODERD(1,  2),CODERD(2,  2),CODERD(3,  2)/10,  3,Z0317/
DATA CODERD(1,  3),CODERD(2,  3),CODERD(3,  3)/10,  5,Z0316/
DATA CODERD(1,  4),CODERD(2,  4),CODERD(3,  4)/ 9,  9,Z0197/
DATA CODERD(1,  5),CODERD(2,  5),CODERD(3,  5)/10,  6,Z0315/
DATA CODERD(1,  6),CODERD(2,  6),CODERD(3,  6)/10,  7,Z0314/
DATA CODERD(1,  7),CODERD(2,  7),CODERD(3,  7)/10,  8,Z0313/
DATA CODERD(1,  8),CODERD(2,  8),CODERD(3,  8)/10, 21,Z0312/
DATA CODERD(1,  9),CODERD(2,  9),CODERD(3,  9)/ 9, 14,Z0196/
DATA CODERD(1, 10),CODERD(2, 10),CODERD(3, 10)/ 8, 12,Z00D7/
DATA CODERD(1, 11),CODERD(2, 11),CODERD(3, 11)/ 7, 16,Z0077/
DATA CODERD(1, 12),CODERD(2, 12),CODERD(3, 12)/ 8, 13,Z00D6/
DATA CODERD(1, 13),CODERD(2, 13),CODERD(3, 13)/ 8, 20,Z00D5/
DATA CODERD(1, 14),CODERD(2, 14),CODERD(3, 14)/ 9, 15,Z0195/
DATA CODERD(1, 15),CODERD(2, 15),CODERD(3, 15)/ 9, 22,Z0194/
DATA CODERD(1, 16),CODERD(2, 16),CODERD(3, 16)/ 7, 27,Z0076/
DATA CODERD(1, 17),CODERD(2, 17),CODERD(3, 17)/ 5, 52,Z0003/
DATA CODERD(1, 18),CODERD(2, 18),CODERD(3, 18)/ 4, 24,Z000B/
DATA CODERD(1, 19),CODERD(2, 19),CODERD(3, 19)/ 6, 31,Z003F/
DATA CODERD(1, 20),CODERD(2, 20),CODERD(3, 20)/ 8, 23,Z00D4/
DATA CODERD(1, 21),CODERD(2, 21),CODERD(3, 21)/10, 29,Z0311/
DATA CODERD(1, 22),CODERD(2, 22),CODERD(3, 22)/ 9, 35,Z0193/
DATA CODERD(1, 23),CODERD(2, 23),CODERD(3, 23)/ 8, 28,Z00D3/
DATA CODERD(1, 24),CODERD(2, 24),CODERD(3, 24)/ 4, 26,Z000A/
DATA CODERD(1, 25),CODERD(2, 25),CODERD(3, 25)/ 2, 18,Z0001/
DATA CODERD(1, 26),CODERD(2, 26),CODERD(3, 26)/ 4, 32,Z0009/
DATA CODERD(1, 27),CODERD(2, 27),CODERD(3, 27)/ 7, 34,Z0075/
DATA CODERD(1, 28),CODERD(2, 28),CODERD(3, 28)/ 8, 30,Z00D2/
DATA CODERD(1, 29),CODERD(2, 29),CODERD(3, 29)/10, 36,Z0310/
DATA CODERD(1, 30),CODERD(2, 30),CODERD(3, 30)/ 8, 37,Z00D1/
DATA CODERD(1, 31),CODERD(2, 31),CODERD(3, 31)/ 6, 33,Z003E/
DATA CODERD(1, 32),CODERD(2, 32),CODERD(3, 32)/ 4, 50,Z0008/
DATA CODERD(1, 33),CODERD(2, 33),CODERD(3, 33)/ 6, 51,Z0001/
DATA CODERD(1, 34),CODERD(2, 34),CODERD(3, 34)/ 7, 39,Z0074/
DATA CODERD(1, 35),CODERD(2, 35),CODERD(3, 35)/ 9, 41,Z0192/
DATA CODERD(1, 36),CODERD(2, 36),CODERD(3, 36)/10, 42,Z030F/
DATA CODERD(1, 37),CODERD(2, 37),CODERD(3, 37)/ 8, 38,Z00D0/
DATA CODERD(1, 38),CODERD(2, 38),CODERD(3, 38)/ 8, 40,Z00CF/
DATA CODERD(1, 39),CODERD(2, 39),CODERD(3, 39)/ 7, 56,Z0070/
DATA CODERD(1, 40),CODERD(2, 40),CODERD(3, 40)/ 8, 46,Z00CE/
DATA CODERD(1, 41),CODERD(2, 41),CODERD(3, 41)/ 9, 47,Z0191/
DATA CODERD(1, 42),CODERD(2, 42),CODERD(3, 42)/10, 43,Z030E/
DATA CODERD(1, 43),CODERD(2, 43),CODERD(3, 43)/10, 44,Z030D/
DATA CODERD(1, 44),CODERD(2, 44),CODERD(3, 44)/10, 45,Z030C/
DATA CODERD(1, 45),CODERD(2, 45),CODERD(3, 45)/10, 48,Z030B/
DATA CODERD(1, 46),CODERD(2, 46),CODERD(3, 46)/ 8, 63,Z00CD/
DATA CODERD(1, 47),CODERD(2, 47),CODERD(3, 47)/ 9, 64,Z0190/
DATA CODERD(1, 48),CODERD(2, 48),CODERD(3, 48)/10, 67,Z030A/
DATA CODERD(1, 49),CODERD(2, 49),CODERD(3, 49)/11, 72,Z0608/
DATA CODERD(1, 50),CODERD(2, 50),CODERD(3, 50)/ 4, 53,Z0003/
DATA CODERD(1, 51),CODERD(2, 51),CODERD(3, 51)/ 6, 55,Z003D/
DATA CODERD(1, 52),CODERD(2, 52),CODERD(3, 52)/ 5, 54,Z0001/
DATA CODERD(1, 53),CODERD(2, 53),CODERD(3, 53)/ 4, 17,Z0002/
DATA CODERD(1, 54),CODERD(2, 54),CODERD(3, 54)/ 5, 19,Z0002/
DATA CODERD(1, 55),CODERD(2, 55),CODERD(3, 55)/ 6, 11,Z003C/
DATA CODERD(1, 56),CODERD(2, 56),CODERD(3, 56)/ 7, 57,Z0072/
DATA CODERD(1, 57),CODERD(2, 57),CODERD(3, 57)/ 7, 58,Z0071/
DATA CODERD(1, 58),CODERD(2, 58),CODERD(3, 58)/ 7, 59,Z0073/
DATA CODERD(1, 59),CODERD(2, 59),CODERD(3, 59)/ 7, 60,Z006F/
DATA CODERD(1, 60),CODERD(2, 60),CODERD(3, 60)/ 7, 61,Z006E/
DATA CODERD(1, 61),CODERD(2, 61),CODERD(3, 61)/ 7, 62,Z006D/
DATA CODERD(1, 62),CODERD(2, 62),CODERD(3, 62)/ 7, 10,Z006C/
DATA CODERD(1, 63),CODERD(2, 63),CODERD(3, 63)/ 8,  4,Z00CC/
DATA CODERD(1, 64),CODERD(2, 64),CODERD(3, 64)/ 9, 65,Z018F/
DATA CODERD(1, 65),CODERD(2, 65),CODERD(3, 65)/ 9, 66,Z018E/
```

```
      DATA CODERD(1, 66),CODERD(2, 66),CODERD(3, 66)/ 9, 92,Z018C/
      DATA CODERD(1, 67),CODERD(2, 67),CODERD(3, 67)/10, 68,Z0309/
      DATA CODERD(1, 68),CODERD(2, 68),CODERD(3, 68)/10, 69,Z0308/
      DATA CODERD(1, 69),CODERD(2, 69),CODERD(3, 69)/10, 70,Z0307/
      DATA CODERD(1, 70),CODERD(2, 70),CODERD(3, 70)/10, 71,Z0306/
      DATA CODERD(1, 71),CODERD(2, 71),CODERD(3, 71)/10,  1,Z0305/
      DATA CODERD(1, 72),CODERD(2, 72),CODERD(3, 72)/11, 73,Z0607/
      DATA CODERD(1, 73),CODERD(2, 73),CODERD(3, 73)/11, 74,Z0606/
      DATA CODERD(1, 74),CODERD(2, 74),CODERD(3, 74)/11, 91,Z0604/
      DATA CODERD(1, 75),CODERD(2, 75),CODERD(3, 75)/12, 76,Z0C07/
      DATA CODERD(1, 76),CODERD(2, 76),CODERD(3, 76)/12, 93,Z0C06/
      DATA CODERD(1, 77),CODERD(2, 77),CODERD(3, 77)/13, 78,Z1809/
      DATA CODERD(1, 78),CODERD(2, 78),CODERD(3, 78)/13, 79,Z1808/
      DATA CODERD(1, 79),CODERD(2, 79),CODERD(3, 79)/13, 81,Z1807/
      DATA CODERD(1, 80),CODERD(2, 80),CODERD(3, 80)/14, 82,Z3006/
      DATA CODERD(1, 81),CODERD(2, 81),CODERD(3, 81)/13, 83,Z1804/
      DATA CODERD(1, 82),CODERD(2, 82),CODERD(3, 82)/14, 86,Z3007/
      DATA CODERD(1, 83),CODERD(2, 83),CODERD(3, 83)/13, 85,Z1806/
      DATA CODERD(1, 84),CODERD(2, 84),CODERD(3, 84)/15, 87,Z6007/
      DATA CODERD(1, 85),CODERD(2, 85),CODERD(3, 85)/13, 80,Z1805/
      DATA CODERD(1, 86),CODERD(2, 86),CODERD(3, 86)/14, 89,Z3005/
      DATA CODERD(1, 87),CODERD(2, 87),CODERD(3, 87)/15, 88,Z6006/
      DATA CODERD(1, 88),CODERD(2, 88),CODERD(3, 88)/15, 90,Z6004/
      DATA CODERD(1, 89),CODERD(2, 89),CODERD(3, 89)/14, 84,Z3004/
      DATA CODERD(1, 90),CODERD(2, 90),CODERD(3, 90)/15, 94,Z6005/
      DATA CODERD(1, 91),CODERD(2, 91),CODERD(3, 91)/11, 75,Z0605/
      DATA CODERD(1, 92),CODERD(2, 92),CODERD(3, 92)/ 9,  2,Z018D/
      DATA CODERD(1, 93),CODERD(2, 93),CODERD(3, 93)/12, 77,Z0C05/
C
      E N D
0              END OF DCEC UPRINT PROGRAM              LINES PRINTED=  1341
```